INFORMATION SCIENCE

Master thesis

Extracting Geographical Semantics from Online News Articles

By: Aleksander Skjæveland Larsen

Supervisors: Bjørnar Tessem, Solveig Bjørnestad

June 1, 2012

Preface

The greatest thanks goes to my supervisors Bjørnar Tessem and Solveig Bjørnestad. You have given me constructive feedback throughout the project, and helped me complete this thesis. The monthly meetings helped setting a pulse for the progress of the project, which have pushed me forwards. I am convinced I could not have finished this project without your help and guidance, so thank you!

Thanks to Terje Hidle, chief engineer in the IT department, for helping with data acquisition from Norge Digitalt. Thanks to the geographer—my wife, Matilde Skår—for patiently answering my questions regarding map data, datums, coordinates and more. You greatly bettered my basal knowledge within your own field, and I gained respect for the complexities of geography. Thanks to the creators of the Oslo-Bergen-Tagger, whom I had a nice email exchange with. Thanks to Eirik Stavelin, who also is using the Oslo-Bergen-Tagger. You had some great ideas regarding usage and how to increase tagging speed, which I shamelessly have stolen in my own library implementation. I would also like to thank my parents, who always have supported me in my studies in Bergen. Also, thanks to my friends in the Master's course, for general sanity upkeep and preventing me from becoming a hermit. And coffee.

Abstract

Several news articles on the web contain geographical locations as significant elements. For the most part, these locations are not available in a format that is machine interpretable. The machine can read in the text of an article, but not derive an understanding of its content. This project aims to find techniques for detecting and extracting locations from the plain text online news articles. The project is limited to articles written in Norwegian, and published in the county of Hordaland. This is done by using methods from design science, for the development and evaluation. A prototype is implemented as a proof-of-concept system using the Clojure programming language. By text analysis, the prototype is able to find mentions of locations in articles. The prototype system have been made available as an open source project and as a Clojure library.

Contents

List of Figures

List of Tables

Listings

1 Introduction

News articles often have geographical locations as significant information elements. Locations are often mentioned by plain text in news articles, relating the article to countries, cities, regions, and more. The sheer multitude of news articles freely available to everyone with an Internet connection, means that automatic processing in order to extract information can be useful for the reader. Unfortunately, news articles on the web rarely have this sort of information in a machine interpretable format, which is required for this sort of processing to be possible. Currently, it is our own understanding of the text that provides us with this information.

Computers are not yet sufficiently proficient in natural language processing, and are not capable of deriving the same set of semantics from a text as we humans are. The more general problem in this context is machine analysis of natural language texts, where the machine is able to understand the semantics in the text. In this project, the effort is focused on a smaller version of the general problem of extracting meaning from text, which is extracting locations. It is the aim of this project to discover techniques we can use to extract geographical semantics from online news articles, and implement these in a prototype system using the Clojure programming language. The focus of the project will be on the Norwegian county of Hordaland, by analyzing articles published in this county.

1.1 Motivation

There are multiple aspects of this project that interest me, and provide motivation to work on it. Some of these motivations are related to the project matter in itself, locations and natural language processing. Aside from this, my own interest in implementing software and the technical challenge required to undertake this project, represents a major motivational influence for me. The prototype system will be implemented in the Clojure programming language, which is running on the Java Virtual Machine (JVM). A technical motivation is—perhaps self evidently—required to undertake an implementation-oriented project. Another motivational force is an interest in semantic technologies and the problems faced in this field.

Semantics used in information systems and on the web, brings information into machine interpretable formats. A number of semantic technologies have emerged over the years, the most visible trend being the rise of the semantic web, popularly dubbed Web 3.0. We can understand the transition with the analogy of going from the old web of documents, to a web of data (Bizer et al., 2008). In the current state of affairs, semantic technologies rely heavily on human users and developers. In order for the machine to gain a semantic understanding, a developer or user must supply the machine with ontologies and valid statements within these ontologies. An ontology is a formal representation of knowledge as concepts within a domain, and the relationships between these concepts. Multiple ontologies can be aligned, in order to achieve correspondence between the concepts.

The system proposed in this project aims to extract geographical semantics automatically by machine analysis alone. I wish to uncover the locations in news articles, in order to use this data in various applications. This means I want to be able to provide machine interpretable data *without* human intervention. The nature of this project is not only oriented towards semantic technologies in themselves. The project is also interested in the utility of the developed prototype system, and how it can be used in different applications. In its most basic form, the functional prototype would take in news articles in the form of plain text, perform processing and return the words that are locations. This output can then be utilized in a number of different settings, which is discussed in section 1.3.

1.2 Research Problem

I want to answer the following research question: "*How can we automatically detect geographical information in online news articles?*". In order to answer this, I will build a software artifact iteratively, as a proof-of-concept system. In order to evaluate the progress and measure the level of success, the following success criteria are proposed:

The system should be able to

1. detect possible locations by text analysis
2. represent complex[1] locations
3. create mappings between an article and locations
4. provide accurate data while minimizing false positives

The geographical scope of the project will be limited to the Norwegian county Hordaland. It is the third largest county by population, with over 490,000 inhabitants. The implementation of the prototype system will be guided by a corpus of articles, collected from online newspapers published in this area. Use of the corpus is discussed in detail in section 3.2. The process of implementing the system will be iterative, where prototypes are produced for evaluation throughout multiple iterations. Within the allotted time, performing the development in three iterations seems reasonable. This is relevant to the first success criterion, since it is restricting the possible number of articles to process.

The second success criterion assumes usage of geospatial map data in the form of shapefiles. A shapefile describes geometries using points, lines and polygons. This data format is much richer than simple coordinate points. For example, where a coordinate point only places a city on a map, a polygon can describe the entire outline of the city. The use of such data is discussed in more detail in section 2.2.3.

The third criterion is concerned with the coupling of articles to the locations that are found. When a location is found in an article, some mapping have to be created to represent the relationship between the location and the article. This is simple if the prototype system return the locations if finds as plain text. It can however become more complicated, if it supports complex locations as in the second criterion.

The fourth success criterion have taken some time to define rigorously. At first, it read "provide correct data", without defining what constitutes correctness. After working with different definitions, I arrived at the current state. Still, a technical foundation was missing. Precisely how would the accuracy of the data be measured? The other criteria

[1] Locations represented by non-primitive shapes like polygons

appear fairly straight forward, as they in a clear way are concerned with the implementation of the prototype system. A measurement of accuracy is less obvious than creating a mapping between article and location. In order to satisfy the fourth criterion, methods from the field of information retrieval (IR) will be used. Specifically, the measurements *precision*, *recall*, and *f-measure* will be applied in order to provide a satisfactory answer to the question of accuracy. Information retrieval, and its use in the project, is discussed in section 3.2.

1.3 Potential Uses

The prototype system to be developed in this project can provide utility within a number of different settings. These are explored and discussed shortly in this section, in order to provide motivation and justification for the project. The prototype system can be exposed as a software library, providing data for a number of different applications.

One potential use is as a plug-in for content management and other publishing systems. Here it would analyze the text before publishing, extract the locations, and suggest geographical tags that can be added. Considering the laborious and relatively menial task of manually typing in tags for articles to be published, this should be a suitable use for the prototype system. This tag generation can be expanded to several other cases where we want some sort of location tags, for example within semantic web. The output from the software could be used as foundation for generating RDF statements according to some given ontology.

Another area of interest is within tool construction that require training data. Using training data, it is possible to construct language processing tools by using frameworks like the Stanford NLP. NLP stands for natural language processing, and is a concerned with the interaction between computers and natural languages, like English or Norwegian. As the focus of the project is on a particular Norwegian region, the prototype system may perform worse on articles published in another region. If so, training data could be generated by the prototype, in order to train a more generalized location detection system.

The software can also be used within generation of statistics and metrics. One could for instance analyze a collection of articles, grouped by news paper, and determine a geographical focus—based on the statistics calculated from data found by the prototype system.

1.4 Project Overview

This thesis document the work done in the project, and an overview is presented here. Following the introduction is the literature chapter. Here previous work is discussed, along with technology considering data and tools. After this, the research method is discussed in relation to the project. Along with the method, evaluation and various limitations to the project are also discussed.

The development chapter contains the three iterations, where the development of the prototype have been documented. Each iteration contains sections on the various problems faced, along with a short summary and evaluation. At the start of the iterations, there is a table providing an overview of the different programming libraries, namespaces and programs used. After the development, the evaluation of the prototype system is performed and the results are discussed. The thesis ends with a summary and conclusion, before suggestion some further work.

2 Literature

Previous work have been reviewed, in order to find relevant literature and technology solutions that can be applied to the project. After discussing the literature, its practical application and the technological choices of this project are discussed.

2.1 Previous Work

The reviewed work is tangent or overlapping the project's area of interest, with some differences. The focus is not on these fields in themselves, but what there is to learn and what is suitable to be used in this project. The first article reviewed is similar in that it also finds locations in text.

The main interest of Fink et al. (2009) is detection of geographical focus in blogs. By analyzing all the posts from a single blog, they attempt to find the location which represents the geographical focus of the entire blog. I noticed that my problem is somewhat different, as I want to detect all locations in a piece of text. In order for Fink et al. (2009) to resolve the overall focus, they do need to detect all the locations—which is where our problem is aligned. For location lookup, they use a gazetteer as the data source.

A gazetteer is a collection of geographical locations, akin to a geographical dictionary. It will typically provide additional relevant data on locations, which may include coordinates, the location's classification, the inclusion hierarchy, population number, and possibly more. The location can be classified as a country, state, city, or as some other meaningful class. An inclusion hierarchy is a hierarchical ordering of nested sets, for example one city is a member of a particular state set, which in turn is a member of a particular country set. Fink et al. (2009) use the gazetteer for location lookup, in order to help resolving the geographical focus. After filtering the blog posts for matches in the gazetteer, population sizes were used for filtering the matched locations. Topological relationships were also used to filter and disambiguate matches.

Disambiguation is the process of resolving ambiguity in the meaning of words. For example, upon retrieving legal documents, it is appropriate to eliminate documents containing

the word *court* as associated with royalty, rather than with law (Ide and Véronis, 1998). Within the scope of finding locations, it is desirable to eliminate words that are used in an inappropriate sense, where the location name may overlap in meaning with other words or locations. The location's inclusion hierarchy was used by Amitay et al. (2004) in disambiguation, in order to determine which location was the relevant one. If one location name matches two different instances of a city, the inclusion hierarchy can help resolve the ambiguity if another relevant region is mentioned elsewhere in the text. For example, a non-ambiguous location might be mentioned, e.g. a region, which may contain one of the ambiguous cities. If so, the ambiguous city names can be resolved by selecting the city that is included within the hierarchy of another location.

Fink et al. (2009) discussed related work by performing a literature review of research relevant to their problem. One of these studies was by Zong et al. (2005), who based their system on a software package named *GATE*. The GATE software was developed for extracting named entities, and is available as a standalone system and a software library. GATE consists of several built-in components, such as a tokenizer, sentence splitter, part-of-speech tagger, and ontology matcher. Most of these tasks represent challenges to overcome in this project, so the software may be of use in my project.

Adida et al. (2011, Chapter 3) discuss automatic annotation mainly in relation to semantic annotation using ontologies. In the important disambiguation step, usage of part-of-speech (POS) taggers are discussed. A POS tagger is software that process text in a given language, in order to perform grammatical tagging. It is mainly concerned with the grammatical and syntactical processing of text. The POS tagger will typically perform sentence splitting, tokenizing, and assignment of tags for parts of speech. These tags will include noun, verb, adjective, and more (The Stanford Natural Language Processing Group, 2012). POS usage seem to be the only directly relevant aspect of the work done by Adida et al. (2011), as they are dependent on use of ontologies.

Amitay et al. (2004) worked with associating geography with web pages, and describe the system they developed to determine a web page's geographical focus. Two approaches for disambiguation are discussed, mainly natural language processing (NLP) and use of a gazetteer. Using NLP, locations are found by the structure and context of sentences and words. The gazetteer approach is often simpler to use, but cannot find locations that are not present in the list. The system they developed is using the gazetteer approach. For

future work, they suggest usage of a POS tagger, while noticing the performance impact this will have on the system. In their survey of previous work, named entity recognition was briefly mentioned.

Named entity recognition (NER) is software that classifies elements in the text into predefined categories. Some examples of these categories can be addresses, personal name, organizations, date and time, quantities, citations, monetary values, and more. We can view the main concern of the NER as adding some semantic understanding to a text—categorization of entities—where the POS tagger mainly is concerned with the syntactical and grammatical understanding. Amitay et al. (2004) and Michael D. Lieberman (2007) mentioned the use of NER software in the survey of previous work, while Amitay et al. (2004) used it in their system.

The approach of Fink et al. (2009) can be broken down in three subtasks: named entity recognition, disambiguation and determining geographical focus. The named entity recognition task uses a NER software to extract locations from the text. It is not clear what software they used or how it was applied. They barely mention the use of the NER, claiming it is "*widely studied and is too broad a topic to review (..)*". Instead of discussing the NER, they focus on reviewing disambiguation and determination of geographical focus in previous studies. They find that use of a gazetteer is common among all the disambiguation strategies. After reviewing the related work, they describe how their system process the blog posts. For each post, the NER is used to extract location entities mentioned in the text. Each entity name is then matched against a gazetteer, which gives them a list of toponyms with coordinates and other relevant data. In order to filter out words that often gives wrong locations, they used a list of *stop places*.

The list of *stop places* is akin to *stop words* in information retrieval, which are common words that have little value in the retrieval process. These are kept in a list and excluded from the vocabulary, in turn giving better results (Manning et al., 2008, p. 27). Similarly, Fink et al. (2009) did this with a list of locations that were adding little value. Both "Obama" and "Coca Cola" were added to the list, as they on occasion got tagged as locations. "Obama" is a city in Japan, and "Coca Cola" is a populated place in Panama, but they turned out to give little value in determining the location.

2.2 Technology

Based on what was reviewed in the previous section, tools and data sources are located and discussed. These will be put into practical use in the implementation of the prototype system. The different technologies are discussed in the following subsections.

2.2.1 Gazetteer

Fink et al. (2009) used GeoNames (2011) as a data source in their research. GeoNames provides data free of charge through web services or as a download. Their database contains over 10 million geographical names. Amitay et al. (2004) use a number of other data sources. One that contains data on Norwegian locations is WorldGazetteer (2011), which "provides a comprehensive set of population data and related statistics". Several of the reviewed articles used gazetteers as a data source. It often serves both as a simple lookup list of names, as well as a method of ranking different locations based on the available data—such as population size and hierarchy. When examining the GeoNames gazetteer, about 55,000 locations are found in Norway.

2.2.2 Part-of-speech tagger

Even though named entity recognition was suggested, used, and referenced in the related literature, I have not been able to locate a Norwegian NER software package. This is unfortunate, as it seems to provide great utility in the projects which have used it, and it seemed relevant to the problems faced in this project. The focus have therefore been on finding a suitable part-of-speech tagger instead.

While working with the literature and searching for Norwegian tools, mainly two approaches are identified in order to obtain a POS tagger. The preferred approach would be to use a ready-made software package, which preferably would be free or open source. There will be two main impediments to this, which is regarding programming language support and Norwegian language support.

The software package may be offered as a library in another programming language. If this is the case, some interface or new language bindings could be constructed. The more likely impediment is the relatively poor selection of tools with support for the Norwegian language. In the literature section, Zong et al. (2005) were using the the GATE software. The usage of this was examined, but it quickly became clear it does not readily support Norwegian language (GATE, 2012). Because of this fact, it is dismissed. If a suitable system cannot be found, it is possible to construct the tagger from scratch, which is the second approach.

In order to construct a tagger from scratch, access to training data is required. A system which supports both approaches is the *Stanford Log-linear Part-Of-Speech Tagger* by The Stanford Natural Language Processing Group (2012). This POS tagger have support for several written languages, and there exists bindings for multiple programming languages. It is also possible to implement support for new languages, which would be required if it is to be used with Norwegian texts—which is not supported out of the box.

However, implementing a POS tagger is probably a project of its own within the field of computational linguistics. Not wanting to implement this from scratch, much effort was spent searching for a suitable solution outside of the literature. A software package was found, namely the *The Oslo-Bergen Tagger* (OBT) by UniComputing (2012). This is a free software package which performs POS tagging on Norwegian texts. It is not tied to any particular programming environment, as it outputs the tagged text directly to the shell. Tools for further processing will be constructed.

2.2.3 Map data

While the gazetteer is a simple coordinate lookup list, modern map data contain a variety of relevant data points concerning geography. A coordinate point consists of a latitude and a longitude, which specifies a zero-dimensional point as a geographical location. The coordinate point is zero-dimensional because it does not have height, length or width. A richer representation of geographical data is the notion of shapes, which is defined by vectors. A vector describes the outline of a geographical entity, or the curvature of a road. Geographical polygons can measure perimeter and area of locations, giving more to reason with than just placement of simple points.

Modern map data may help with geospatial reasoning, using a more complex representation of locations. This representation alone does not give any data, that helps differentiating between geographical instances, like cities, villages, and regions. If a coordinate based data set contains a location with 3 inhabitants, it is identical to a location with 3 million; they are both a simple point. Gazetteers may help in this respect, as they can provide population numbers, which can be used in conjunction with the geographical shapes to compare the locations. If both population data and a polygonal representation of a city is available, population density could be calculated and used to rank locations.

In order to hold geographical data, a suitable database is required. An ideal candidate is PostGIS, as this database support different geographical features. PostGIS is an extension to the PostgreSQL object-relational database. Using PostGIS, it is possible to query based on coordinates and distance, which can support some of the challenges faced in this project. Another use for the database is to hold the mappings generated between the online news articles and the different locations, as well as other relevant data.

2.2.4 Statens Kartverk

The number one provider of geographical data from Norway is Statens Kartverk (the Norwegian Mapping Authority), a public agency under the Ministry of the Environment (Andersen, 2009). The University of Bergen participates in the project Norge Digitalt (Digital Norway), which is "the Norwegian government's initiative to build the national geographical infrastructure" with the aim to "enhance the availability and use of quality geographical information among a broad range of users, primarily in the public sector" (Kartverket, 2011). Because of this participation, I am able to attain high quality map data from Statens Kartverk that normally would be prohibitively expensive to use. It is fortunate for me, but a blow regarding social aspects, since one normally has to purchase access to the raw data.

As Statens Kartverk is a public agency, it is fair to say their largest source of income is taxpayer money. Critics have argued they should make their data free to the public, reasoning that the public already have paid for it. Another argument for free map data is the success of free weather services, a notable example being Yr.no[1].

[1] http://www.yr.no/

Yr.no gives their data away for free, both to end users through their web site and to developers through an API. This helps Yr.no gain market share, without developing applications for the plethora of devices available. They are able to do this with the from external developers that make applications using the free data. Note that these developers are probably not motivated to help Yr.no, but to make useful applications. This results in benefit for the end users, who are able to choose from a wide variety of applications for their devices. The result for Yr.no is making their services more widespread than they would be able to achieve by developing applications themselves.

Free data is an asset both to the consumer and business entrepreneurs: a report from the European Union suggests values totaling 400 billion NOK can be created by releasing free public information in Europe (Noer, 2008). Brenna (2008) argues the money that Statens Kartverk earn from sales is negligible in the grand total. In 2007 their total income was 757.1 million NOK, with 10 million from sales of analog products and 24.6 million from sales of digital products. Brenna argues it is naïve to expect a public agency to perform as well as top business people, and increase sales to a meaningful level.

In 2009 it was announced that Statens Kartverk would release their maps for free use. This was met with great anticipation, but also skepticism considering the ongoing criticism. As it turned out, the release only concerned pre-rendered maps, and not any underlying data. Technical oriented critics want the actual data released, not mere pictures. With only pre-rendered maps, one is not able to search for locations names, get coordinates and shapes in machine readable formats, which is what is needed for technological innovation using map data (Solstad, 2009).

2.2.5 Norge Digitalt

As described in the previous section, the University's participation in Norge Digitalt have provided this project with geospatial data. This data was delivered as shapefiles, a commonly used format for geographical data. The different file formats included is listed table 1, with a description from Esri (2009). The files are specific to the shapefile standard, which can be exported to the PostGIS database.

Upon examining the data received from Norge Digitalt, the map data has been provided in three different groupings: N50, N250 and N500. These names corresponds to different resolutions used by Statens Karverk[2]. The N50 data is in the scale 1:50 000, the N250 in 1:250 000, etc. The data is split into separate collections by municipality, in nested folders with each data set contained in an archive file. For each resolution, there are unique directories for over 30 municipalities, all of which containing files that need processing. Not wanting to traverse and process this relatively large directory tree manually, a custom tool should be constructed to extract and import this data into the PostGIS database.

Filetype	Description
shp	shape format; the feature geometry itself
shx	shape index format; a positional index of the feature geometry to allow seeking forwards and backwards quickly
prj	projection format; the coordinate system and projection information
sbx, sbn	a spatial index of the features
xml	Metadata for ArcGIS; stores information about the shapefile

Table 1: File formats that constitutes the Norge Digitalt map data

2.2.6 Name data

Both the full names of people and other geographical locations are interesting data to use in this project. Name data of geographical locations can be regions, counties and municipalities, and they may exist in poorer data sets that does not contain coordinates. Using these data, it is possible to construct simple lookup lists, which can be used in order to perform filtering. Municipality names in Norway are public data, and is gathered from norge.no[3], a portal to public information.

Lists of personal names can be used in order to disambiguate locations. A major source of ambiguity when it comes to finding geographical locations, is the naming convention of using geographical locations as surnames.

[2]http://www.statkart.no/nor/Land/Kart_og_produkter/Kartdata/
[3]http://app.norge.no/kart/kommunerifylke/

The overlap of surnames and locations is normal in Norway, but this differs around the world. If a similar project is done in for example Iceland, surnames will be a different problem—perhaps even not one at all. Icelanders use a patronymic (sometimes matronymic) naming scheme, identifying the immediate father (or mother), not using family names. Problems might still arise even here, as of immigration and people with deviating surnames. Patronymic naming schemes exists all over the world, so a similar prototype system in another language will have to take this into consideration.

Statistisk sentralbyrå (Statistics Norway, SSB) have published a list of surnames used by 200 people or more. This list is collected and put in a database for use by the prototype system in disambiguation of possible location names. The purely list lookup approach has some limitations. For example, the personal names collected only represent a statistical sample of names in active use. This means the collection only contains names commonly used today. The selection is cut off at 200 people, meaning the collection do not contain a name if only 150 people use it. The names we miss out on, may include rare variations or plain deviations in spelling: "*Trond*" and "*Trånn*", the former a common male name, the latter an extremely rare spelling variant with the same pronunciation. Some names are not present in the collection because they have become unfashionable or inappropriate. Names of traitors and dictators often fall out of active use, but might still be useful in analysis—depending on what types of text to process.

Some imported foreign names have become statistically common, such as "*Ali*" and "*Singh*". There might still be names that one would want in the collection, which does not have sufficient mass to show often enough statistically. The Chinese surname "*Ng*" comes to mind, which is not found in the SSB data sets. While such foreign names does not represent Norwegian locations, it is still desirable to find them; people with foreign names might have a Norwegian middle name. In addition to this, finding instances of foreign names in a text will still help the disambiguation process, even though there does not exist an overlap with Norwegian locations.

Some of the names with smaller syntactic differences actually have statistical representation, and is present in the collection: "*Christer*" and "*Krister*"; "*Katrine*", "*Kathrine*", "*Katherine*" and "*Cathrine*"; "*Mohammad*" and "*Mohammed*". An extended name recognizer could implement some of these common patterns, in order to recognize names not already in the list. In addition to this, there could be support for fuzzy name match-

ing. For example, typing "*Alexander*" one also want to match "*Aleksander*". Working with these problems in depth seem to be more related to building a name recognizer, rather than a location recognizer. The focus in this project will be on getting good enough name recognition for disambiguation use, and not work extensively on name recognition for its own sake.

2.2.7 Programming Language

I will use the programming language Clojure[4], which is a dynamic language that targets the Java Virtual Machine (JVM). There is also support to target the CLR and JavaScript platform. Clojure is a Lisp dialect, which enables interactive development and provides a powerful macro system. It is mainly a functional language, with several immutable, persistent data structures. Targeting the JVM provides excellent interoperability with Java. Clojure compiles to JVM byte code to make Jar files. This enables me to use Java libraries from Clojure, and in the opposite direction, providing Clojure projects as Java libraries.

The Clojure community have a central repository for open source libraries, akin to Maven in the Java community. This is the Clojars[5] repository, where nearly all Clojure libraries are found. I choose to work in Clojure because of the great community, platform support in the JVM, and because I like the flexibility of a Lisp language.

[4]http://clojure.org/
[5]https://clojars.org/

3 Method

For this implementation oriented project, using the research method of design research is a suitable match. Design research is also referred to as design science in the literature. The method contains several guidelines, which is discussed in relation to this project. The evaluation will be performed using a descriptive method, as well as an analytical one. After reviewing the research method, some limitations are discussed.

3.1 Design Research

The reason for design research being a natural choice for implementation oriented projects, is because it allows us to build new systems in order to perform evaluations. This is justified in the following quote:

> "(...) without research efforts directed toward developing new solutions and systems, there would be little opportunity for evaluative research." (Nunamaker Jr and Chen, 1991)

Hevner et al. (2004) suggest seven guidelines for use in the design science research process. They advice against mandatory use of the guidelines, and stress that it is the individual researcher that must determine when, where and how to apply each of the guidelines in a specific research project. The guidelines are listed here, to discuss and determine their relevance in the project:

1. Design as an artifact
2. Problem relevance
3. Design evaluation
4. Research contributions
5. Research rigor

6. Design as a search process

7. Communication of research

The *first* guideline—*design as an artifact*—will be at the core of the research project. I will answer the research question by building a proof-of-concept system as an artifact resulting from the research process. The research process will—for the most part—in some sense be concerned with or related to the development of this prototype system. Preparations with tools and data will be done in order to support the implementation, and the evaluation will also be closely related to it.

The *second* guideline, considering *problem relevance*, is also relevant for the research question. As discussed, uncovering semantics automatically is a problem, and it is hard to perform without human intervention. This project tackles a specific case of the larger issue of semantics, by focusing on geographical semantics. By focusing on this subset, the scope of the project is limited to an appropriate unit of work within the time allotted. There is a solid justification in this guideline, arguing for development of the prototype system.

The *third* guideline considers the *evaluation of the design*. Various design evaluation methods are listed, which can be applied to software artifacts in the research process. The methods suggested are *observational*, *analytical*, *experimental*, *testing*, and *descriptive* evaluation. I will use a descriptive method, and construct detailed scenarios around the prototype system to demonstrate its utility. Hevner et al. (2004) write that "descriptive methods of evaluation should only be used for especially innovative artifacts for which other forms of evaluation may not be feasible". By using descriptive evaluation, I am able to explore the prototype's utility for a number of different uses, outside the scope of other evaluation methods. In addition to this, I will also be using an analytical evaluation method throughout the development. This will be of the type dynamic analysis, and will determine the prototype system's performance. I will apply the dynamic analysis by using techniques from the field of information retrieval, which is discussed in detail in section 3.2. Using the metrics of precision, recall and f-measure, I can evaluate the prototype system based on how it performs on a corpus of news articles.

My main *research contribution* is the prototype system I will develop, which will attempt to provide a possible solution to an unsolved problem. As Hevner et al. (2004) state, the "criteria for assessing contribution focus on representational fidelity and implementability". My aim is to provide an instantiation of an artifact, and document how it was developed. The purpose of this is to enable other researchers and developers to implement a similar solution, using both the prototype system and the research process as a foundation. The prototype system can by itself be a starting point for further development, or as a guideline for a completely new system. The various problems to be faced throughout the research process is documented and discussed, as these can be helpful in designing similar solutions.

Research rigor is represented both in the methodology in which the prototype system is developed, as well as in how it is evaluated. The end result is a working piece of software, which is evaluated quantitatively using the metrics from information retrieval. The research process will be documented and discussed thoroughly. Using both analytical and descriptive analysis, I will show that the prototype system is applicable to the problem, and generalizable to the problem domain.

The *sixth* point is viewing the development process as a *search process*, in order to discover an effective solution to a problem. This guideline can be related to the project since I will be developing the prototype system iteratively, while using information retrieval metrics to evaluate the performance to guide development. This can be related to the generate-test cycle Hevner et al. (2004) describes, by implementing the artifact in order to test it. However, I do not consider this guideline an integral part of the project. I accept the relationship between iterative development and the generate-test cycle, but will not apply the guideline to the project.

Regarding the final point, *communication of research*, I primarily want to reach the technically oriented research audience. The technology and techniques to be discovered, are interesting mainly for other researchers or developers that can build upon what is found. The resulting prototype, implemented as a proof-of-concept system throughout the research process, will not be a fully mature software package. This means it will not be in a state acceptable for use in real world applications. Considering this, my research will not need to be communicated to a wide audience outside the intended one. When—rather if—the software matures to a point it is ready for use in an applied setting, a wider au-

dience might find it interesting. Primarily, it is this thesis that will act as a medium for communicating the findings to the proper audience. The prototype system will nevertheless be available as a starting point for further development.

3.2 Evaluating with Information Retrieval

Information retrieval (IR) is the field of study concerned with searching for documents, and information within documents. The documents can be of an unstructured nature, usually text, without a semantically apparent structure. IR can be used to filter document collections or further processing of a set of retrieved documents (Manning et al., 2008).

IR has some important performance and correctness measures, which will be used in this project. These are the metrics *precision*, *recall*, and *f-measure*. These numbers will provide some level of accuracy, from the prototype system's performance on a corpus of news articles. Manning et al. (2008) define precision as "the fraction of retrieved documents that are relevant", which is expressed in equation 1.

$$Precision = \frac{relevant\ items\ retrieved}{retrieved\ items} = \mathbb{P}(relevant|retrieved) \tag{1}$$

Recall, defined as "the fraction of relevant documents that are retrieved", is expressed in equation 2. It is trivial to achieve a high recall, simply by returning all the documents in a query. In my case, I could simply classify every word as a location, in order to score a high recall. If I do this, the precision will suffer.

$$Recall = \frac{relevant\ items\ retrieved}{relevant\ items} = \mathbb{P}(retrieved|relevant) \tag{2}$$

Precision and recall are measures that trade off each other. It is necessary to perform the search or classification in a way that eliminates the erroneous result, giving higher precision while maintaining recall.

The combination of these metrics can be evaluated with the f-measure, expressed in equation 3, which is the weighted harmonic mean of precision and recall.

$$F = \frac{1}{\alpha\frac{1}{p} + (1-\alpha)\frac{1}{R}} = \frac{\left(\beta^2+1\right)PR}{\beta^2 P + R} \qquad \textit{where} \qquad \beta^2 = \frac{1-\alpha}{\alpha} \tag{3}$$

In these formulae, P represents precision and R represents recall. Van Rijsbergen (1979) explains that "this measures the effectiveness of retrieval with respect to a user who attaches β times as much importance to recall as precision." The default *balanced F-measure* weights precision and recall equally. This means either making α = ½ or β = 1. I will use β = 1, which will simplify the formulae to what is expressed in equation 4.

$$F_{\beta=1} = \frac{2PR}{P+R} \tag{4}$$

In order to evaluate the prototype with these formulae, a corpus of data is needed. The corpus should be constructed to cover a wide range of different news articles, which should include different problems and challenges. Implementation-wise, the prototype system should not be directly dependent on the corpus. It should not be over-specialized on a given corpus, as this defeats the purpose of finding general techniques and ideas.

Extra data should not be added to the data sets, if only to tweak and tune the software's performance on the given corpus. An example of this would be the use of lists containing personal names. These lists should be acquired from a source, and treated as a *black box*. In this example, I should not add new names to the lists when the prototype returns false-positives when processing the corpus. There will likely be special cases when the spelling differs that are not represented in the lists. Also, rare and new names might not have been added yet. It might be tempting to do add such data, in order to improve the rating of the prototype system. This should be avoided, as it will lead to over-specializing the prototype to the corpus. Over-specialization to one particular data set will not add any real value in the software, when it is performing on real world data. Adjusting the data in this way is intellectual dishonesty, and is to be avoided.

3.3 Limitations

Before the development is started, much thought have gone into considering issues and problems that can arise. These concern the relevance, applicability and utility of the project. The limitations are discussed in the following sections.

3.3.1 Source Code and Licensing

Upon reading research articles, I noticed that source code is rarely provided. There exists justifications both *for* and *against* providing source code with research projects. If proprietary data or systems are being used in projects, releasing the source code can be problematic. In other cases, it can even be in the best scientific interest to not release the source code. This is due to the fact that every software system has its bugs. By allowing reuse of the code, there exists a greater chance for the bugs to live on, if the code is not scrutinized by whomever uses it.

If the code is used without being reviewed, bugs are less likely to be found. In contrast, by only describing the algorithms, you force others to reimplement the system. Previous bugs will not automatically be carried on, but new bugs can (and will) be introduced. In certain cases, this aspect may be very important in order to achieve a high degree of academic rigor. In this implementation oriented project, I do not think keeping the source code closed has any merit. My particular sentiment in this case, is that buggy software is better than no software. Providing the source code may lead to the software being utilized, as ideas and techniques can be fully understood from the code. The argument of replicating bugs can be used in the opposite direction, claiming instead that by making the source code available, other people can help finding and fixing bugs instead of starting from scratch.

In order to encourage reuse and further development of the prototype system, an open source license will be applied to it. The software should be usable in other projects, so the license should not be too restrictive. To allow reuse in a wide variety of settings, in addition to protecting further development, I will apply a weak copyleft license to the prototype system. This license class is commonly used for software libraries, as it allows

linking and redistribution without requiring the new software to be distributed under the same license. Only direct changes to the library itself is required to be distributed with the same weak copyleft license. I will use the Eclipse Public License (Eclipse Foundation, 2011), which is a weak-copyleft license. This enables others to use the software as they see fit, even in proprietary settings. The code will be hosted on GitHub[1], which kindly have provided a free educational account for this project. GitHub provides web-based hosting for software development projects that use the Git revision control system.

3.3.2 Social Aspects

There is a possibility the performance of the prototype system will depend on what data sources are available at the time of implementation. Some data sources are freely available and open for public use, others require compensation or have restrictions on usage. Because of this, it is not possible for everyone to use commercial data sources. It will also be problematic to release the prototype system as an open source project if it is coupled with a proprietary data set. This may carry social implications for the prototype system, if it only operates on such data sources.

Hopefully everyone will be able to benefit from the findings that will be made, regardless of social aspects and access to commercial data sources. I hope to discover techniques that do not depend on a specific data source, but generally applies to those available.

3.3.3 Data Sources and Copyright

One issue to consider is the activity of crawling and storing data from the web. Content providers online usually have restrictions on usage of their material, and the content is protected under copyright law. Some providers may provide free data as a service, through an API, e.g. Yr.no.

In order to download the weather data as an XML feed, there are some guidelines and restrictions one will have to comply with. These specify how often one can fetch data, and how one may store and use it. The reasons for such limitations are both technical

[1] http://github.com/

and commercial. Fetching data very frequently is not allowed, as this puts too much load on the provider's servers. Usage is often restricted to non-commercial applications, as the providers have their own commercial interests in the data they serve. Such regulations are something everyone using data sources on the web will have to consider, as conventions and law might differ among providers and countries. If the web site does not offer the data as a service, there are often restrictions as to how much data that can be stored.

Considering a search engine, the entire content of a web page have to be downloaded and analyzed in order for the search engine to perform the indexing. Some search engines even offer cached versions of the site, meaning that the entire page is stored in the search engine's database. This might be in direct violation of copyright law and usage terms, but still happens. This legal gray area will have to be explored where such a project is to be undertaken, if one wishes to stay completely within the boundaries of law. The software developed in this project will at some point have to download entire web pages in order to perform the analysis. However, the content should not be offered directly to the user.

3.3.4 Language Barrier

As the prototype system may rely heavily on specialized parsing for Norwegian texts, there might arise a language barrier inherent in the software. Because of this, the overall utility of the software can be reduced. A consequence can be that international users will not find it useful as a library, since they probably do not need to analyze Norwegian texts. The software can perhaps prove useful only as a guideline or inspiration in implementing something similar. Taking these aspects into account, I want to minimize the effect of a language barrier as much as possible. My interest is in finding general techniques and ideas that can be implemented in multiple languages. The corpus used in this project will be a set of Norwegian articles from online news sites. I limit the scope of this project to analysis of Norwegian articles, as it is my first language. I certainly have more insight into naming conventions in Norwegian than I would in any other language, without being a domain expert by any measure.

3.3.5 Working Conditions of the Prototype

As the main focus is to find techniques and explore ideas, I must take care to focus on the research aspect of the project rather than a large code base. The resulting artifact from the research process will be a prototype, not a production ready system. As this is a Master's project, the focus will also be on the scientific contribution by developing and evaluating the prototype system within the scope of design research methodology. To demonstrate the prototype's utility, I will make one or two demonstration applications to show some potential uses. Ideally, the software will mature to a degree where it is possible to provide it as a library, so it may be used in other projects.

4 Development

The main body of development work was done in three iterations. The early parts of the development phase consists mostly of exploratory programming and work done in data processing and conversion. Throughout the iterations, the main parts of the project was separated out as separate libraries—with most of this exploratory, data processing and transformation code omitted.

This means that the final version of the prototype system, which have been released as a library, does not contain all the code that is discussed in the iterations. An example of this is the SOSI parser (described in section 4.1.2), which is not used in the final library. In order to give access to the omitted code, a separate branch have been introduced to the git repository. It is available with the main extraction software from Github[1]. Use the `git checkout` command on the branch `thesis-dev`. This will give an unpolished, but complete access to all the code that is referenced in the discussions.

When a part of the prototype system is being discussed, the relevant location in the `thesis-dev` repository will be referenced with namespace listings. A namespace represents the location of a file, in the Clojure project. Consider the following namespace:

Namespace: `ogrim.parsers.sosi`

This points to the file `sosi.clj` located in the folder `src/ogrim/parsers/`. Where relevant, this will be listed at the start of the development sections. A table is provided at the start of each iteration, which gives an overview of the different programs, libraries and namespaces used and discussed in the current iteration. These different parts are explained when they are introduced in the text, so the tables are not used as a replacement for a proper explanation and discussion. However, if it should be desirable to look up a definition after it has been introduced, it might be faster to consult the table instead of scanning through several paragraphs of text.

[1] https://github.com/ogrim/clj-egsiona

4.1 First Iteration

The first tasks worked on was the installation of tools, getting familiar with technology and data acquisition. Focusing on getting PostGIS up and running and learning about geospatial data took a large portion of my time, in addition to working with the data from Norge Digitalt. A major challenge was understanding the data and find ways to apply it in the prototype system. The details of the implementation performed in this iteration is discussed in the following sections.

Name	Type	Description
ogrim.parsers.shape2postgis	Namespace	Norge Digitalt extraction tool
ogrim.parsers.sosi	Namespace	SOSI parser
news-crawler	Program	Article collector for corpus construction
Enlive	Library	A selector-based (à la CSS) templating and transformation system
clj-egsiona.corpus	Namespace	Corpus construction tools
clj-obt	Library	Interface to the Oslo-Bergen-Tagger

Table 2: Programs, libraries and namespaces used in first iteration

4.1.1 Data Processing

Namespace: `ogrim.parsers.shape2postgis`

This tool is concerned with importing the geographical data into a system that can be used from the prototype. For this purpose I have used the PostGIS spatial database. Due to the nested nature of the data from Norge Digitalt, I wrote code to automatically extract, convert and insert all the data into PostGIS. This code can be useful for anyone that needs to process similar map data.

Recursive extracting of archives containing the map data was performed with a custom bash (a Unix shell) script. It traverses directories recursively, extracts and then deletes the archive files. When the raw data had been extracted, the directories were traversed to find

all the relevant shapefiles to be inserted into the database. Only the `.shp` file, described in table 1, needs to be specified in the command for the PostGIS insertion.

A function takes each of these files and generates database statements, one set for preparation[2] and another for insertion. These statements are very similar, the only difference being a parameter denoting the preparation-only with no insertion. This was done because the database had to be prepared first, in order to avoid any errors upon insertion. An example of a generated preparation statement can be seen in listing 1.

```
shp2pgsql -p -s 32632 -W LATIN1
  /home/ogrim/data/N50-N5000_Kartdata/N50_Kartdata
  /fylke_kommune/12_Hordaland/1228_Odda/UTM32_Euref89
  /Shape/1228_Arealdekke_lin.shp N50_Kartdata.arealdekke_lin |
  psql -h localhost -p 5432 -d norge-digitalt -U postgres
```

Listing 1: Example of PostGIS preparation statement

There was generated one set of statements for each shapefile, in this case totaled 458 for the N50 resolution, 374 for N250 and 360 for N500. By storing the statements in a single file, they were executed one at a time with a bash command. The end result is a database successfully populated with the geographical vector data available.

4.1.2 Central Place Name Registry

Namespace: `ogrim.parsers.sosi`

After working with the vector data imported in the previous section, some observations regarding its applicability was made. Although geographical vector data can help in retrieval and geospatial processing, it did not offer data like location names—data needed for this project. There was very little meta data included in the first batch of geographical data from Norge Digitalt, so I inquired further and gained access to the Sentralt Stedsnavnregister (central place name registry).

[2] creating tables and schemas

The Sentralt Stedsnavnregister (SSR) is the official registry of location names. It is administrated and distributed by the Norwegian Mapping Authority, but is proprietary data. The SSR data set was also received from the Norge Digitalt project. The data set was delivered with the same folder structure described in section 2.2.5. In addition to this, there was a single file containing all the location names from every municipality. The data format was not shapefiles, but a distinctively Norwegian format named SOSI. This is an acronym for "Samordnet Opplegg for Stedfestet Informasjon" (Coordinated Approach for Spatial Information). Since 1987, SOSI have been developed and used by the Norwegian Mapping Authority. Being a distinctively Norwegian data format entails little, or no free tools to handle conversion of the data.

Due to the obscurity of the SOSI data format, custom tools had to be constructed to extract the relevant data. Fortunately, the format is simple to parse because it is represented by plain text. The SOSI format consists of nodes, an example of this can be found in listing 2. The number of full stops in the front of the words denotes hierarchy, followed by node name and some optional data.

```
.TEKST 260999:
..OBJTYPE SSRForekomst
..NAVNTYPE 146
..KOMM 1259
..DATAFANGSTDATO 20110919
..OPPDATERINGSDATO 20051228
..SSR
...SSR-ID 1052969
...SNAVN "Dale"
...SNREGDATO 20051228
...SNFOREK VVEKA
..NØ
6753968 -54058
```

Listing 2: Example of SOSI node

A SOSI file is parsed by reading through it line by line, taking care to be efficient and avoid holding on to lines unnecessarily in memory, because there are 3.7 million lines in the largest SOSI file. For each `TEKST` node that is found, it is parsed by a function that extracts the data and emits a Clojure data structure. The nodes that contain the relevant data is `KOMM`, `SNAVN` and `NØ`. The SOSI node in listing 2 is parsed to the data in listing 3.

```
["Dale" 6753968 -54058]
```

Listing 3: Parsed SOSI node

Parsing the SSR data for Hordaland resulted in 265,029 locations. Several of these are duplicates, where all the fields are identical. These were easily filtered down to 94,700 distinct locations. The data set still contained a lot of duplicate instances, of which none were completely identical. The problematic locations had mostly identical fields, with only the coordinate points being different. Upon examining some of these locations, it became clear the coordinate points varied between 50 and 1000 meters. It is possible to filter duplicate locations based on distance, but I wanted to avoid setting an arbitrary limit on the minimum distance allowed between locations. Instead, the notion of one location name per municipality was applied. This filtered down the remaining locations to 73,671 unique instances. Compared to the approximately 55,000 locations in the GeoNames data set, which includes all of Norway, 73,671 is a very large number for a single county.

My interest having been sparked by writing this tool, I looked into more formalized techniques of parser construction, where I found alternative approaches. In retrospect it seems to have been more efficient to use parser generator tools, instead of writing the parser from scratch. A formal way to do this is writing a BNF grammar, which the parser generator uses to automatically construct the required parser. BNF stands for Backus Naur Form, and is a notation technique for describing context-free grammars (Wirth, 1976, Chapter 5). There exists modern variants to parser generating, like ANTLR (ANother Tool for Language Recognition) by Parr (2007).

4.1.3 Article Collector

To facilitate corpus construction, I needed a tool to scrape articles off relevant news sites. This was implemented very early in the iteration, and is available as a standalone project at github[3]. The name is *news-crawler*, even though it is in fact a scraper. At first, it was supposed to enter online news-sites and automatically find and extract the articles. However, I only needed it to construct the corpus, so it became mainly a scraper. There were

[3]https://github.com/ogrim/news-crawler

several concerns in my mind while building it. For one, I wanted to avoid hammering or flooding the sites being scraped. This was achieved by using random pause intervals between each page download. Another Master's student got his IP address banned when scraping data for his project, thus my concern with this (Ruben E. Oen, pers. comm.).

In order to extract the article content, a system of matching HTML nodes was required. One should never parse HTML with regular expressions alone, as this is a surprisingly irregular format in practice (or in the wild). This is to say: web browsers are *very* forgiving when it comes to correctness of the HTML. This makes it hard to parse regularly, so a more generalized system is useful. For this I used Enlive, a selector-based templating and transformation library for Clojure. In addition to regular expressions, Enlive gives great flexibility to declare filters with ease and little code. This was done so the tool could be directed at different online news sites without the need for large changes. The *news-crawler* tool was used to collect articles for the corpus, which have been tagged manually with the corresponding locations.

4.1.4 Corpus Construction

Namespace: `clj-egsiona.corpus`

Using the article collector over a period of time, 250 articles were collected from the 4th to the 11th of November, 2011. The online news sites used was *Bergens Tidene*[4] and *Bergen-savisen*[5].

An estimated 100 articles should be in the final corpus, with some overhead for reduction and selection. The newspapers used are quite similar in content because of their overlap in geographical focus. Furthermore, I also anticipated a large number of the articles to be more or less duplicates; articles based on press statements and police reports.

Some custom helper functions were written to support the manual tagging of the articles, for example `next-article` and `insert [tag]`. When calling `next-article`, the article is printed to console and an id counter is incremented. This ensures the tags inserted

[4]http://www.bt.no/
[5]http://www.ba.no/

with `insert [tag]` refers to the relevant article. These helper functions enabled me to work quickly and with ease, directly in the console, without moving my hands off the keyboard. This helped me complete the tagging within one working day. When performing the tagging, if the article was a near duplicate of a previous one, I would simply call `next-article` which skipped the current article and dropped it from the corpus. Some articles were also dropped on the basis that they had the wrong geographical focus, for example articles regarding the economic situation in the EU region. Very few articles had no geographical focus, but these were consequently dropped.

In the end, the 250 articles had been reduced to 113, with a total number of 593 tags. The metrics—precision, recall and f-measure—was implemented in code, ready to be run by a single command. This made it possible to continuously run the evaluation throughout the development phase, to get instantaneous feedback on the effect of both smaller changes and entirely new functions.

4.1.5 Oslo-Bergen-Tagger

The library developed for interacting with the *Oslo-Bergen-Tagger* was named `clj-obt`, and have been made available as a standalone project at Github[6] and from Clojars.org[7].

Due to restrictions imposed by the JVM, it is not possible to change the working directory after the JVM is started. Hence, I needed a method of calling the tagger independent of the location where the prototype system is run from. Unfortunately, the launch script that came with the tagger only works when calling it from the tagger program directory. This makes it impossible to call the tagger from the working directory of the prototype system, which is fixed at runtime. A workaround would be installing the tagger directly in the project directory, but this is a hack to be avoided.

Upon inspecting the tagger's launch script, I found that it is basically one rather complicated bash command—and not a proper script. A representation of how it works can be found in figure 1, with the different program names in the main boxes. The script pipes together four commands, calling different programs in sequence and passing the

[6]https://github.com/ogrim/clj-obt
[7]http://clojars.org/clj-obt

resulting output forward. It takes an input file and pass it to the *mtag* multitagger program, which is a tokenizer, morphological analyzer, and compound analyzer. Then it uses *vislcg3* to perform constraint grammar tagging. If statistical disambiguation is selected, it will use *OBT-Stat*[8] to perform the disambiguation. Finally, a small *perl* script controls printing of the results, but this is not an integral part of the tagger. This script contains relative paths to most of these programs, which is a problem since the working directory cannot be changed in the JVM.

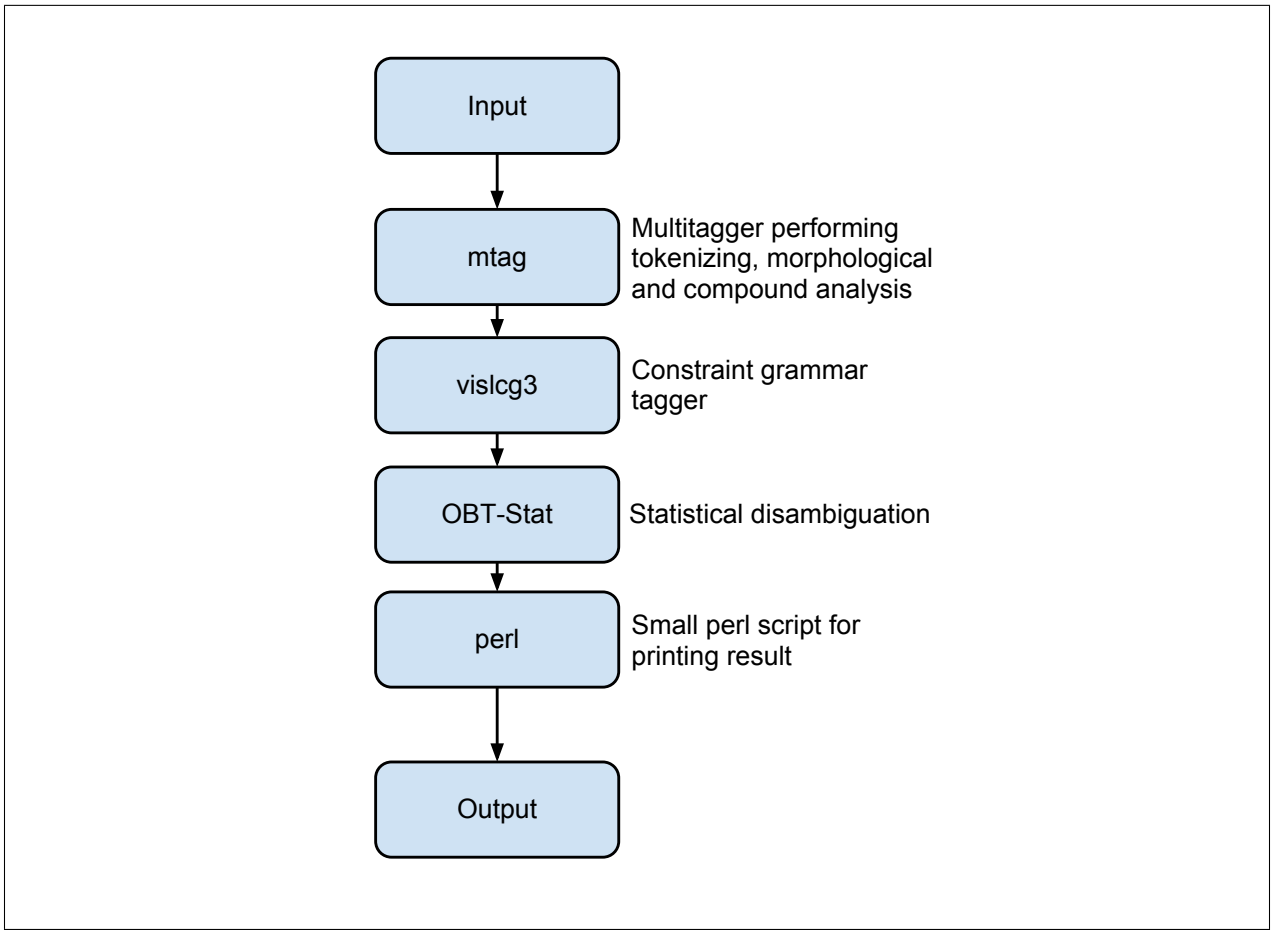

Figure 1: The Oslo-Bergen-Tagger Model

The solution arrived at, was to create a function that emits the required script in a generalized manner. This is possible by inserting fully qualified paths to the tagger program into the script. As the script will need to be called from the command line, it is outputted from the function to a temporary file. The Clojure function that generates this script, can be seen in listing 4. There are three instances of `obt-path`, which are replaced with the full

[8]https://github.com/andrely/OBT-Stat

path to the tagger program directory. The temporary file is then made executable when the `clj-obt` library is initialized, as you would with any other executable file on Linux.

```
(defn- script-content [obt-path]
  (str "#!/bin/sh\n"
       obt-path "/bin/mtag -wxml < $1 | vislcg3 -C latin1 --codepage-input utf
          -8 -g "
       obt-path "/cg/bm_morf-prestat.cg --codepage-output utf-8 --no-pass-
          origin -e | "
       obt-path "/OBT-Stat/bin/run_obt_stat.rb | perl -ne 'print if /\\S/'"))
```

Listing 4: Function wrapping the original OBT-script with full path

When the tagger could be called programmatically, focus shifted to processing of the output. The format the tagger emits, seen in listing 5, seems somewhat akin to XML.

```
<word>For</word>
"<for>"
        "for" prep
<word>14</word>
"<14>"
        "14" det fl kvant
<word>dager</word>
"<dager>"
        "dag" subst appell mask ub fl
<word>siden</word>
"<siden>"
        "siden" adv
```

Listing 5: Output from the Oslo-Bergen-Tagger

The original tagged word is enclosed in `<word>`-tags. In the next line the lowercase version of the word is its own tag, the only problem being it is not a proper tag; it never closes—and it is enclosed by quotation marks. What follow appears to make more sense: lemmas and grammatical tags. In this example the lemma is identical to the original word, which at first glance might seem like unnecessary duplication. This does however have its reasons, as the lemma of a word can differ from the original form. In addition to this, if you skip the disambiguator, there might be several lemmas with different meaning, as seen in listing 6.

```
<word>vold</word>
"<vold>"
        "vold" subst appell mask ub ent <<<
        "volde" verb imp tr1 <<<
```

Listing 6: Example of tags with different lemmas

The parser reads the tagged output from OBT, and constructs a data format more usable for Clojure—using native data structures. Each word gets its own map, with keys that maps to the data values. A vector holds the tags. The data in listing 5 is thus parsed into the data format seen in listing 7.

```
[{:tags ["prep"], :lemma "for", :word "For", :i 1}
 {:tags ["det" "fl" "kvant"], :lemma "14", :word "14", :i 2}
 {:tags ["subst" "appell" "mask" "ub" "fl"], :lemma "dag",
  :word "dager", :i 3}
 {:tags ["adv" "<<<"], :lemma "siden", :word "siden", :i 4}]
```

Listing 7: Data from the Oslo-Bergen-Tagger parsed to Clojure data structure

A bottleneck in tagging performance seems to be related to calling the OBT program. In my experiments, I found that tagging an article of 1100 words took 1.5 seconds, while tagging the four words "*For 14 dager siden*" took almost 1 second. My computer specifications are not important, rather the time difference in the example—indicating startup time as relatively slow. Some domains might require better performance from the tagger than mine. A problematic domain would be processing of short texts in large volumes, since this would require the tagger to restart very often. A possible improvement to the tagger itself, could be to make it work in some sort of server or daemon mode. It would then run non-stop and would not have to be restarted for each piece of text it tags. This change would require access to the source code, and without the code I do not know if this is a feasible solution.

Another approach to reduce the impact of the startup time would be to tag more text each time OBT is called. In discussion with a Ph.D. student, who also is using the Oslo-Bergen-Tagger, I learned he was tagging multiple texts at once to get better performance (Eirik Stavelin, pers. comm.). This can be achieved by concatenating multiple pieces of text, such that OBT tags everything in one single invocation. For each piece of text that is

concatenated, one can discount the startup time from the total processing time—adding in the overhead of tagging a larger volume of text.

Taking the concatenation idea into account, this was implemented in the library. The final implementation of the `clj-obt` library is described by the system model in figure 2. This diagram describes the data flow in the library, and shows the difference when using the concatenation feature.

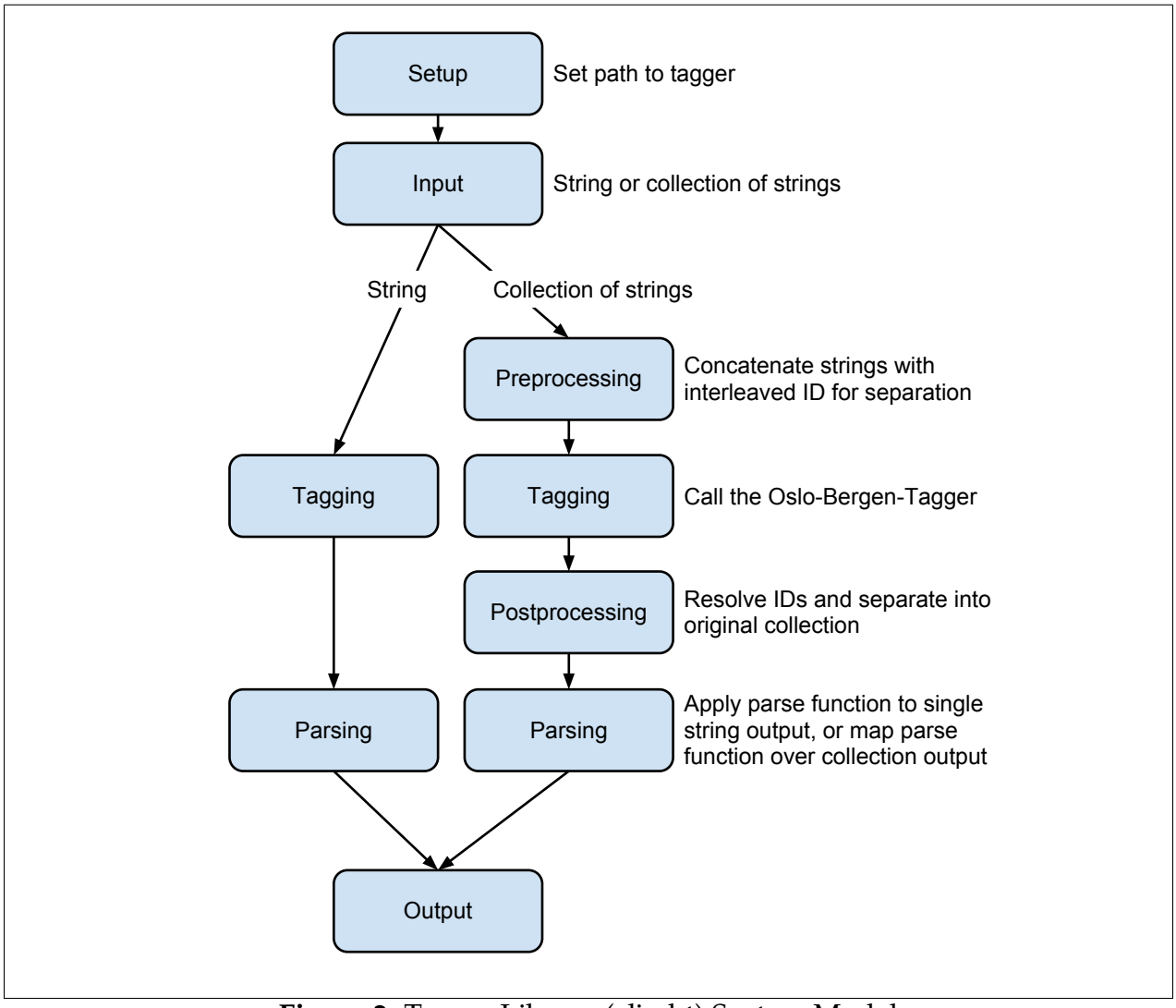

Figure 2: Tagger Library (clj-obt) System Model

After some email exchange with the original authors of the *Oslo-Bergen-Tagger*, my library was added to their website[9] and Github[10] page. Hopefully, this library will be of use to others—not only within this project.

4.1.6 Changing Premises

Throughout this first iteration, the premises of the project shifted as I failed to utilize the geospatial data to a meaningful degree. After working with the spatial data and technologies, it seems I made little progress on solving the core problem of this project. The work done to this point led me far in understanding spatial data, but my knowledge is still too basal to be utilized to a satisfactory degree. Although I understood much of the theory around map data—like coordinates, datums and the technology involved—I faced problems putting this knowledge into practice. At the current level of experience, I found it hard to solve many of these problems, some of which had no apparent solution. Clean textbook examples are usually simple to understand, but real life problems tend to be a little more confusing and less neatly defined. The amount of time spent to get to my current level of knowledge seem to be too high, considering the problems still remaining. It became clear to me I had to reprioritize and use my time more effectively.

I made a list of functionalities needed in the prototype and remaining work, as seen in listing 8. This was then prioritized based on what would deliver the most value for the software's utility. Notice the `[A]`,`[B]` and `[C]` markings, which indicates priority. The items are marked with `DONE` as they are completed. After it became clear an item would not be completed and should be dropped, I marked this with `CANCELED`.

[9] http://tekstlab.uio.no/obt-ny/lastned.html
[10] https://github.com/noklesta/The-Oslo-Bergen-Tagger

```
* [A] fikse navnehåndtering
* [A] bruk navn i disambiguering
* [A] sjekk om lokasjoner er duplikater med genitiv s
* [A] forbedre navnesjekk ved å bruke addresse-gjenkjenning
* CANCELED bruker geocoder og fylker
* DONE [A] jobbe med gramatik
* DONE [A] skrive utfyllende om artikler
* DONE [A] skrive om kart
* [B] Lage demoapp til presentasjon
* DONE [B] skrive om datakilder
* DONE [C] fiks forsiden
* DONE lage tabell med data til evaluering
* CANCELED skriv spørring for intersecte geonames og ND
* CANCELED intersecte SSR og ND
* CANCELED regne ut TD-IDF
```

Listing 8: Excerpt of todo items from my planner, an overview of priorities

This list helped in figuring out that I was spending too much time on secondary functionality, and helped me identify my most important priorities. Some of the lower priority features had been worked on for a long time without providing utility in the prototype system. These features would not be properly utilized unless the basic functionality of the prototype system were in place, primarily the location detection. I consider location detection, even without geospatial processing, to be more important than other features. To match this importance, I had to focus my efforts here instead. The prototype system must first be able to perform text processing before applying the additional techniques and data sets. I consider the use of spatial data to be secondary, mainly to be used for improvement of the disambiguation process.

4.1.7 Evaluation

This iteration have been hectic, informative and educational. It has also helped me steer the progress of the project and focused my efforts. Much time went into obtaining the data sets required, working with tools and building knowledge. There were some faulty assumptions, which impeded my progress. These were related to the applicability of the geospatial data obtained. A considerable amount of work went into processing this map data, before determining that I needed additional data—preferably with location names. These were acquired fairly quickly, as I now had a contact within the correct department

at the university. It was problematic that this data was in the SOSI format, with little tool support to be found online, but a parser was constructed.

From the literature review, I learned that a POS tagger was desirable for use by the prototype system in processing news articles. When I obtained a proper Norwegian tagger, a solid foundation for use within the development environment was built—in form of a library. This library—`clj-obt`—is continuously being improved as new requirements, bugs or quirks are discovered. The construction of the corpus was performed satisfactory with ease and speed, much thanks to the tooling support constructed early on.

In retrospect, I have identified some areas where use of existing libraries and tools would have saved me some time. In the article collector, time could have been saved by using a HTML scraping library instead of processing the content on my own. Some time after the `news-crawler` tool was constructed, I found the Java library *boilerpipe*[11] which "provides algorithms to detect and remove the surplus 'clutter' (boilerplate, templates) around the main textual content of a web page". The approach I would prefer now, would be to construct my own software to find article nodes on the main page, while depending on boilerpipe to extract the content. I also discussed an alternate approach for the SOSI parser, by writing a BNF grammar to use with a parser generator. These were not the largest impediments, which turned out to be problems regarding geospatial data and technologies. This in turn forced me to revise my approach to the problem.

In summary, this iteration was not executed according to my initial expectations. I had some problems and challenges, but I also succeeded and made progress. Too much time was spent on simply acquiring the data, which took quite some time in the larger organization the University of Bergen constitutes. Time went into email correspondence—or attempts thereof—finding the correct people and establish contact. When I at last found the right—and very helpful—people, everything went smoothly as far as data acquisition was concerned. This effort was helped greatly by my advisor, Bjørnar Tessem, who made some phone calls. When I finally had obtained the data, a large amount of work went into determining the applicability to the project, which was hard to determine in advance due to my lacking experience. This could perhaps been avoided by better planning and research into the spatial data formats ahead of time, or by cooperating with domain specialists.

[11] http://code.google.com/p/boilerpipe/

There were also parts of the iteration that went smoothly, or at least with less problems. The foundation built for using the Oslo-Bergen-Tagger in my programming environment have been a great asset. This was implemented as a library, usable for others outside the scope of this project. Corpus construction was also completed in a timely fashion. Perhaps the most important lesson from this iteration is the adjusted expectations regarding use of high-grade map data, as this proved more challenging than first anticipated. I have discussed the problem faced, and how they were handled; changing focus away from geospatial data—and to functionality that more quickly provides value and utility in the prototype, which will be the focus of the next iteration.

4.2 Second Iteration

The transition to the second iteration was not as clearly defined as it may appear in writing. The largest problem from the first iteration was the challenges regarding geospatial data. This is discussed extensively in sections 4.1.1 and 4.1.6. The problem existed in parallel for some time throughout parts of the second iteration as well. Keep in mind that I at first had given the geospatial data use a lower priority, rather than dropping it completely. Throughout the iteration, it got more and more postponed as I focused on more important functionality. This resulted in the geospatial features being put on hold indefinitely, as I saw there would not be enough development time available.

In practice, I transitioned into the second iteration naturally as I gained ground on my initial impediments. When my domain knowledge and the libraries matured, development focus shifted from tooling to the actual problem at hand: location detection. Most of the development work in this iteration is done to support the disambiguation process, in order to improve the precision rate. This have been a very important aspect of the prototype, since it is trivial to score a high recall rate. An overview of the programs, libraries and namespaces used and discussed in this section is available in table 3.

Name	Type	Description
clj-egsiona.country	Namespace	Detecting geographical entities from lookup lists, like counties, regions, municipalities, etc.
clj-egsiona.geocode	Namespace	Client to geocoding library
geocoder-clj	Library	Interface to geocoder service
clj-egsiona.profession	Namespace	Recognition of professions
clj-egsiona.address	Namespace	Detecting addresses
clj-obt	Library	Interface to the Oslo-Bergen-Tagger
clj-egsiona.name	Namespace	Detecting names

Table 3: Programs, libraries and namespaces used in second iteration

4.2.1 Finding Locations

In order to find words that are locations, a method of selecting candidate words was needed. The candidate words are in turn filtered and disambiguated further, with the goal to achieve the best possible results, i.e. a high precision and recall rate. A simplistic method of selecting candidate words is by capitalization alone. Capitalized words may denote a name of some kind—person, location or business—or it might just be the first word in a sentence. However, relying on capitalization alone can be the source of several problems.

There are many instances where the capitalization will be used incorrectly, not counting typographical errors. One example of this is some online news articles that have subheadings throughout the text. In some articles, I have observed subheadings written out in all capital letters. It is of course simple to detect capitalization, but it is more challenging to determine the semantics; is an all caps sentence really a subheading? The proper way to mark up a subheading in the source HTML, is to assign the appropriate level of the H-tag, e.g. `<h3>`, and use CSS to achieve the desired styling of the text. This leads to separation of the content data and presentation, which is good practice. If done properly, it is very simple to filter out any text contained in H-tags. It is therefore more problematic when the markup is used in a convoluted manner, by mixing the presentation and content to achieve stylistic effects.

In addition to high coupling between data and presentation, irregular use of markup is a problem semantically. By not using the proper H-tags, the authors are not conveying the actual meaning of the particular piece of HTML. It is for instance not helpful to blind users that rely on screen reading software. The software reading the source HTML might interpret and read the content wrongly, if the source code is not correct. Deeper discussion of these issues belong in a project on interaction design or an accessibility study, so the extent of my interest in these problems are limited to the processing of such texts.

To find possible locations, it is possible to rely on capitalization alone if no other means of processing the text are available. Having access to a POS tagger, it is only natural to use the output from this. The tagged text from the POS tagger is split into words, with sentence separators.

In the prototype system, some functions are able to process the article without using a selection of candidate words. This include functions that rely on lookup lists, like address detection (discussed in section 4.2.7) and personal names detection (discussed in section 4.2.5. Other functionality which does requires a selection of candidate words, will find the locations by disambiguation. One particular way selection of candidate words was done, have been by simplistic use of grammatical processing, discussed in the following section.

4.2.2 Candidate Words with Simplistic Grammatical Processing

To find possible locations, a method of selecting candidate words was needed. This was achieved with some simplistic grammatical processing, which is described in this section. At first, a coarse selection of candidate words was attained by capitalization alone. This had some problems, and a better method of selection was required. By examining the output from the prototype system, false-positives could be identified. It became clear some features never could be locations, like typographical markings and symbols.

The problematic tags were put in a list, and filtered away in the selection process. This removed all instances of commas, parenthesis, dashes, etc. Still having a number of erroneous words left, this list was expanded with additional tags that never seemed to represent locations. Examples of these are verbs, prepositions, pronouns and conjectures.

It is true the typographical markings could be removed without the help from a POS tagger, but the same is not true for the other word categories. Without the POS tagger, this would be problematic to achieve in a generalized manner. The prototype system does not need to encode the meaning of words, as the words are grammatically classified by the POS tagger. Filtering away unwanted tags is a simple, but valid way to use the POS tagger output.

Processing based on simple filtering only gives a precision of 24.1%, recall of 95.2% and a f-measure of 36.9%. The recall is good, but this is trivial as returning all words would give 100%. The f-measure is important here to give a balanced view between precision and recall. It is weak, due to the poor precision. Precision currently seems to be the challenge, so the focus should be on improving it.

4.2.3 Geographical Entities from Lookup Lists

Namespace: `clj-egsiona.country`

The use of lookup lists of names was discussed in section 2.2.6. In this section the focus is on lists of various location names. Examples of the data in these lists are countries, counties, municipalities, and more. Following the list construction, functionality for querying the lists was implemented. It is possible to find all mentions of locations in a given text, where all matches are returned. Alternatively, it is possible to perform a boolean query, to find whether a string is a location, i.e. exist in the lists.

Consider querying for countries. To find all mentions of countries, the `find-countries` functions takes in a string, and returns a list of all the matches. The predicate operates similarly, and gives a boolean answer to whether a string matches an instance in the country list. The `country?` function takes a string and returns true or false. In total, this namespace has data on, and is checking for, instances of countries, continents, regions, counties and municipalities.

Not present in the `clj-egsiona.country` namespace, is the SSR data set which also can be thought of as a lookup list. Since SSR contains all location names in Hordaland, it should be able to be used in disambiguation. This would probably help the precision rating

substantially. Support for this was implemented in the `clj-egsiona.core` namespace, and is being used to filter out possible locations not found in the data set. This improved the precision, but the downside is tying the prototype to the proprietary SSR data set.

Upon performing the evaluation, the prototype scored 55% precision, 77.6% recall and 62% f-measure. This is better than the last f-measure of 36.9%, at the cost of worse recall.

4.2.4 Geocoder

Namespace: `clj-egsiona.geocode`

In the prototype system, there are mainly two important uses for a geocoder. One is as a part of the disambiguation process, where the geocoder can be queried to find whether a possible location actually exists, while the other is to get geographical data on location names. In order to access a geocoder service, I used a library developed for this purpose: `geocoder-clj`, available at Github[12] and from the Clojars[13] repository. This library is an interface to several service providers, such as Google, Bing and Yahoo.

The default geocoder provider in the library is Google, so this was used without change. Some terms of usage are present in the *Google Geocoding API*, which was desirable to comply with. Among others, they restrict usage of the service to 2500 API calls per 24-hour period. Misuse of this service will result in temporary or permanent banning, which I of course wanted to avoid (Google, 2011). An interface layer was implemented between the `geocoder-clj` library and the consuming code in the prototype system. This interface layer restricts the usage by the limits Google have set. It does this by keeping track of requests per 24-hour period, and limit the available API calls accordingly.

In section 4.2.3, the SSR data set was used as a filtering device in disambiguation. When doing this, it simultaneously confirms the existence of a location and pairs the possible location with a coordinate point. This coordinate can then be used for retrieval purposes. Because the SSR data set is restricted and proprietary, an alternate method of pairing locations to coordinates should be found.

[12] https://github.com/r0man/geocoder-clj
[13] http://clojars.org/geocoder-clj

After the geocoder support was implemented in the prototype system, I worked with various techniques to use the result from querying the geocoder service. Unfortunately, this proved to be problematic. Because of similar location names all over the world, the scope of the lookup needs to be limited to a certain geographical region. This would be a feature that is dependent on what functionality the service provider offers in the geocoder. If not an explicit feature, the limitation can be worked around by concatenating the search string with a restricting country or region clause. The geocoder is then limited to looking up locations within the restricting region, reducing the scope of the search. In turn, this lead to another issue, because the geocoder seem reluctant to return empty result sets. If a location does not exist, the geocoder will return data on the restricting location instead. Considering the geocoder call in listing 9.

```
(geo/geocode "festplassen")

=>

({:city "Havndal",
  :country {:name "Denmark", :iso-3166-1-alpha-2 "dk"},
  :location #geocoder.location.Location{
    :latitude 56.6538525,
    :longitude 10.1971161},
  :street-name "Festpladsen",
  :street-number nil,
  :postal-code "8970",
  :region "Central Denmark Region",
  :provider "Google"})
```

Listing 9: Call to geocoder and response

The geocoder is queried for "*festplassen*", a location in the city of Bergen, in Hordaland. The geocoder returns a response with data on a location in Denmark, which have been matched in the `street name` field. However, consider the geocoder call in listing 10, where a restrictive clause is concatenated with the original query: "*festplassen, hordaland*".

```
(geo/geocode "festplassen, hordaland")

=>

({:city nil,
  :country {:name "Norway", :iso-3166-1-alpha-2 "no"},
  :location #geocoder.location.Location{
    :latitude 60.2733674,
    :longitude 5.7220194},
  :street-name nil,
  :street-number nil,
  :postal-code nil,
  :region "Hordaland",
  :provider "Google"})
```

Listing 10: Call to geocoder with restricting phrase, and response

The geocoder apparently returned a positive match as a result from the query. Notice the general lack of data in the fields like `city`, `street name` and `postal code`. This seem to indicate it has matched "*hordaland*", instead of "*festplassen*". When checking the coordinates in an actual map, the point is placed directly in the center of Hordaland county. This property make the geocoder unsuitable as a simple filtering device in itself.

In order to use the geocoder for matching possible locations to proper locations, the query result can be compared to any data already in possession. Alternatively, the lack of data can also be informative, as in listing 10 where several of the fields are empty. Different approaches to use the geocoder data in disambiguation was explored, for example dropping instances where the city field was empty. Some attempts were made to use a restricting clause, like a region. In all instances, the use of the geocoder for disambiguation gave worse results. It was therefore taken out, and is currently not in use by the prototype system.

4.2.5 Personal Names

Namespace: `clj-egsiona.name`

Another useful feature in disambiguation is to identify personal names. It is discussed in section 2.2.6 that surnames often overlap with geographical locations, so this functionality is to be expected.

The proposed solution for this problem requires access to lists of names, preferably both given names and surnames. When such data is available, identification is quick and easy. While processing the possible locations, one simply queries the name database in a boolean manner, determining whether a word is a name. In my solution, I have lists of names stored in hash sets, allowing for quick lookup. There are 1089 given names and 3317 surnames available.

Identification of middle names have proved a little more ambiguous, as middle names tend to overlap with both given names and surnames. A person might have a proper given name as middle name, others have proper surnames. The total number of names are may also vary, with some people seemingly having two middle names. In the writing, middle names are sometimes abbreviated to a single capital letter. In order to properly identify a middle name, I first check if a word is a given name or a surname. If neither are true, I fall back to check the capitalization, which will allow for abbreviations, e.g. "*Aleksander S. Larsen*".

The algorithm that combines all these functions, starts by detecting the surnames. Next step is expansion of the name in its entirety. This is done by recursively looking up possible middle names in reverse, so a *surname-to-given-name* expansion of the entire name is achieved. When a name cannot be expanded further, the algorithm will perform validation and reduction in the forward direction. This ensures that the first word in the expanded full name is a valid given name, since the word will be dropped if it is not valid.

After working with this functionality, evaluation of the prototype systems gives a precision of 62.6%, recall of 78.2% and f-measure of 67.4%. This is an improvement from using SSR in the filtering, which had an f-measure of 62%.

4.2.6 Professions

Namespace: `clj-egsiona.profession`

While investigating the false-positives from the prototype system, I noticed that several professions were reported as locations. Thinking of the *stop words* list from IR, a similar construct was used by Fink et al. (2009): *stop places*. I consider that some types of entities can be filtered out by a similar approach, in this case with a *stop professions* list.

The profession detection uses two approaches to match words. It looks for specific professions, for example "lawyer" and "project manager". It also looks at common suffixes indicating a profession, like "officer", "chief", "doctor", etc. Currently there are a mere 3 specific professions, and 9 suffixes.

This comes close to over-specialization to the corpus, so an attempt was made to implement the functionality in a generalized manner. The professions detection can be extended with additional professions *without* considering the false-positives from the system. This is in contrast with a traditional list of stop words, which is only extended *after* finding problematic words in the results.

4.2.7 Addresses

Namespace: `clj-egsiona.address`

Some of the articles contain addresses in different forms. An assumption is made that an address consists of at least the street name, followed by the optional street number. Street names have certain properties which enables detection by simple processing. They often have common suffixes, which indicates some sort of category describing the location. For simple streets, there are multiple variations in English: *road*, *drive*, *avenue* and *boulevard*. Where streets intersect, we can find names that include *corner* and *square*, and sometimes even an *alley*. There exist numerous such examples, and collecting these for use in address recognition is one approach. In Norwegian, we do not use word-division in the same manner as in English. This is due to different rules regarding construction of compound words. For example, consider the street name "Abbey Road". For the words themselves,

"abbey" can be translated into "kloster", and "road" into "vei". However, the street name "Abbey Road" would be translated into "Klosterveien" in Norwegian; a compound of both words as well as changing the article of "vei" to the definite form "veien".

Two approaches have been identified for using knowledge of Norwegian addresses in the prototype system. One approach is to look for common suffixes in single words. Another one is to look for certain stand-alone words that denote a street name. These approaches can be used in combination. By stand-alone words, I refer to words that are not part of compound street names. These stand-alone words are often preceded by personal names, for instance "Magnus Barfots gate". Sometimes street names are shortened, e.g. "*gate*" (street) to "*gt.*" (st.). These special cases are only accepted if followed by a number, as this represents a stronger indication of an actual address. If one were to look for these special cases in every word in a text, normal words at the end of sentences would also be matched, e.g. "Vi reiste lan**gt.**" (we traveled far). Because the translation of this example does not make sense, a meaningful example in English would be "We traveled fa**st.**". Of the non-compound words, some examples include *smau* (alley), *haug* (hill) and *dal* (valley).

Using a list of street names[14], I manually found many of these common suffixes and put them in a list—totaling 75 entries. This list is then used by several functions to detect possible streets. After exploring various ways of using address detection for disambiguation, while fixing bugs and quirks, the f-measure had risen to 73.9%.

4.2.8 Evaluation

While the first iteration largely was spent working with tools, data and getting familiar with the domain, this iteration have seen more work on the core problem of the prototype system. Formal evaluation with information retrieval metrics was started in this iteration. Throughout the course of the development, the progress was closely monitored by the metrics calculated from executing the prototype system on the corpus. This have proved very useful, as it guided the development process by immediately revealing the effect of newly implemented functions and features.

[14] http://no.wikipedia.org/wiki/Liste_over_Bergens_gater

After working with finding potential locations—selection of candidate words—some simplistic grammar processing was applied. This lead to an f-measure rating of 36.9%, which left lots of room for improvement on the behalf of precision, which was only 24.1%. Lists of data was introduced, with personal names and geographical entities like countries, regions, municipalities, and more. These were used for various lookup function for the disambiguation process. After applying the SSR data set as a filtering device, the f-measure had risen to 62%.

Usage of a geocoder was explored, with hopes for useful functionalities, like location validation and resolving coordinates. The prototype system performed worse with the geocoder, due to various problems. A major problem was when querying with a restrictive clause, where the geocoder seems reluctant to return empty results. The geocoder was therefore taken out of active use.

Usage of a list of personal names was discussed and explored. Support for detection of middle names was implemented, and the functionality performed well. The f-measure was increased to 67.4%. This was followed by a discussion of address detection, and the problems faced in that respect. Using a list of street names, common suffixes were found and put into a list. After implementing address detection in the disambiguation the f-measure was at 73.9%. Not discussed in detail are mere bug fixes, and smaller problems, which further improved the f-measure to 74.9%.

In summary, the metrics have gradually been improved throughout this iteration. Only recall has suffered, but this started at a high of 95.2%. Through all subsequent measurements the precision and f-measure was improved. Several different avenues for disambiguation was explored, such as address detection, personal name detection and geocoder lookup. In the end the geocoder was taken out from disambiguation, because the metrics got worse when it was used.

4.3 Third Iteration

This iteration's focus have mainly been on finalizing the prototype system, making it runnable and easy to configure. At the start, the focus was still on improving the metrics, by working on disambiguation and candidate word selection. This effort was then halted, in order to complete the system. By exposing the prototype system as a library, its programmatic applicability have been tested by implementing a demo application. Not only is this useful for the descriptive evaluation, but it also serves the purpose of demonstrating the utility of the prototype to an audience. By having a concrete demo application, this utility can be communicated with greater clarity and efficiency.

Name	Type	Description
clj-egsiona.domain	Namespace	Domain name recognition
clj-egsiona.entity	Namespace	Entity recognition
clj-obt	Library	Interface to the Oslo-Bergen-Tagger
clj-obt-service	Program	Tagger Web Service
clj-egsiona.tagger-provider	Namespace	Implementation of client to web service tagger
clj-egsiona	Library	The Extraction Software as a separate library
clj-egsiona.core	Namespace	Main entry point for prototype system
clj-egsiona.processing	Namespace	Contains functions related to processing and disambiguation

Table 4: Programs, libraries and namespaces used in third iteration

4.3.1 Grammatical Processing

In the previous iterations, only simplistic grammatical processing was used. For example, a list of unwanted tags was kept and used to filter the tagged text. At this point, more sophisticated processing have been explored. Instead of filtering away unwanted tags, I wanted to select candidate words directly.

When analyzing articles tagged by OBT, I noticed that prepositions often appear before locations. This can be used to select possible locations, with subsequent disambiguation to be performed. Locations are rarely followed or preceded by verbs, although we can find examples of the contrary: "*Bergen vokser i omfang*" (Bergen grows in size). This rule is not set in stone, as language is very flexible. It does however seem to be used rarely, so I assumed words before verbs are not locations. If a valid location should erroneously be filtered away by this rule, it might still be referenced multiple times throughout the text. Hopefully this possible location is used in a sentence, in a manner so it is not filtered away with the verb. After selecting candidate words based on the grammatical tags from OBT, problem cases are filtered away. This include words that are numbers, contains less than two characters, in all capital letters (per subheading problems discussed in section 4.2.1), not starting with capital letter, and if the next word is a verb.

Replacing the previous location-finding functionality with this method, the f-measure got slightly worse to 73% from 74.9%. There was quite a substantial drop in precision, which went from 73.1% to 64.5%. The f-measure did not suffer a greater degradation, because the recall was increased 10.8%, from 82.5% to 93.3%. If only the f-measure is to be considered, these results show that the previous method of selecting candidate words was better. But analyzing the other metrics, it became clear that the precision can be improved with better disambiguation. The recall rate cannot be improved if the potential location is filtered away earlier in the process, so it is desirable to keep these locations and focus on further improvement of the precision.

All aspects of the grammatical processing discussed in this section, may give the impression that the implementation was a neat, sequential process. In fact, several parts of the prototype system was developed in parallel, or in chunks in between one another. After implementing the grammar selection just described, some effort was focused on different functionality to help in disambiguation. This lead to implementing the functionality described in section 4.3.2, which in turn was incorporated in the grammatical disambiguation that belongs in this section. Chronologically, the grammatical disambiguation belongs in its own section after the entity recognition section, but content-wise it is too short to justify this. This explanation is now—ironically—longer than the actual content:

Disambiguation of the grammatical word selection is fairly simple. Given names are filtered out, along with domain names and other entities. A more interesting problem

is nouns in the possessive case that cause false-positives. These words are sometimes locations, but are problematic to report as such. Often, the prototype system will find a location both in a neutral form and in the possessive case. It is desirable to only keep the neutral form, so if the last character of the word is an "s" and the word is tagged as possessive, it is filtered out.

4.3.2 Entity Recognition

Namespace: `clj-egsiona.domain`
Namespace: `clj-egsiona.entity`

The listed namespaces contains functionality to detect different types of basic entities. These are problematic words, often being miss-tagged as locations. It is not completely akin to a stop word list, as discussed in section 2.1, but an expansion in the disambiguation part of the software. A list of stop words does necessarily not in itself convey any meaning as to what a given stop word constitutes. In the prototype system, I am trying to build a foundation for a generalized and extensible entity recognizer. Since there are some form of semantics involved, it can be extended without directly considering false-positives from the tagging process, as in section 4.2.6 with the professions recognition.

The domain name recognition is trivial, as it is just checking for known top-level domains (TLD) at the end of words. For the moment, there is a small selection of 7 TLDs, but this can easily be extended. This functionality removes both URLs and email addresses.

The functionality discussed in this section is also at risk of becoming over-specialization, if not done in a specific and generalizable manner. The main entity detection, which lives in the `entity` namespace, have the most in common with a list of stop words. It is the perhaps least specific functionality, because it is a collection of problematic nouns. These nouns are in conflict with other functionality detecting locations, for example address detection. Recall in section 4.2.7, where address detection was discussed, common suffixes of street names are put into a list. These are also causing trouble with ambiguous words that is giving false-positives, which the entity recognition tries to disambiguate. For example "plass" (place) is a suffix in the address recognizer that leads to several false-positives: "parkeringsplass" (parking lot), "skoleplass" (school yard), "byggeplass"

(construction site), and more. The detection of problematic words was found by examining the output from the prototype system, which seem closely related to constructing a list of stop words.

The more specialized part comes in with an attempt to introduce grammar handling in the entities listed. In Norwegian, the article vary according to the gender of its noun. When an entity is found, it is put into a list representing the noun's gender in the indefinite form. The nouns are then automatically expanded into the different forms, from the indefinite to the definite article, both in singular and plural forms. An example of this expansion can be seen in listing 11. Currently, only male and neuter words are supported, but there is a separate list for irregular nouns that must be expanded manually upon insertion.

After this functionality has been incorporated into the disambiguation processes, the prototype system was evaluated. Precision had increased to 69.1% from the last low of 64.5%. The recall changed slightly to 93.6% from 93.3%. This gives an f-measure of 76.8% which have been the highest yet.

```
male:      "konvensjon" -> "konvensjonen" "konvensjoner" "konvensjonene"
male:          "avtale" -> "avtalen" "avtaler" "avtalene"
neuter:    "direktorat" -> "direktoratet" "direktorater" "direktoratene"
irregular: "senter" "senteret" "sentre" "sentrene"
```

Listing 11: Automatic grammatical expansion of nouns

4.3.3 Tagger Web Service

This project is available as a standalone project at Github[15] and on the Clojars[16] repository. A system model is provided in figure 3.

After a longer bus trip while working in the Windows installation on my laptop, I decided the situation regarding tagger platform dependency had to be resolved or somehow alleviated. The Oslo-Bergen-Tagger only supports Linux and Mac, and the `clj-obt` library only supports Linux. If only the source code for the multitagger used in the OBT system

[15]https://github.com/ogrim/clj-obt-service
[16]https://clojars.org/clj-obt-service

would be made available, a Windows version could probably be compiled. This could potentially lead to supporting Windows in `clj-obt`.

Corresponding with the authors of OBT over email, I learned the release of the source code had stalled due to copyright issues in parts of the code. After inquiring for the possibility of obtaining the source code, I was given positive signals for this to happen. This effort unfortunately stalled, and I decided the fastest route to use the tagger in Windows would simply be to implement a web service to expose the functionality through. This would allow use of the prototype system on a wider range of platforms. If the prototype is used on a non-Linux system, the tagger service can be hosted on a dedicated Linux machine somewhere on the network. Alternatively, it possible to use desktop virtualization and host a Linux installation on a Windows or Macintosh computer.

Clojure made it very simple to implement both the server and the tagger library interfacing, demanding a mere 33 source lines of code. I choose to only support `HTTP POST` requests, as `GET` requests are not suited for longer parameters. The text that will be sent to the tagger can potentially be very long, which could cause problems. The server is simple to run, because it was compiled to a standalone jar file. This requires only a Java installation, with no knowledge of Clojure required. To start the server, simply issue one command with the desired port number and the tagger location, for example as in listing 12.

```
java -jar clj-obt-service-0.0.3-standalone.jar 8085 /home/ogrim/bin/The-Oslo-
   Bergen-Tagger
```

Listing 12: Command to start tagger web service

Figure 3 is a model of the `clj-obt-service` system, describing how it is built. It can return plain text or the json data format, which is selected by sending the `POST` request to the appropriate endpoint. The service will then call the tagger on the input data, and perform necessary encoding before returning the result. The client that consumes this web service is discussed in the following section.

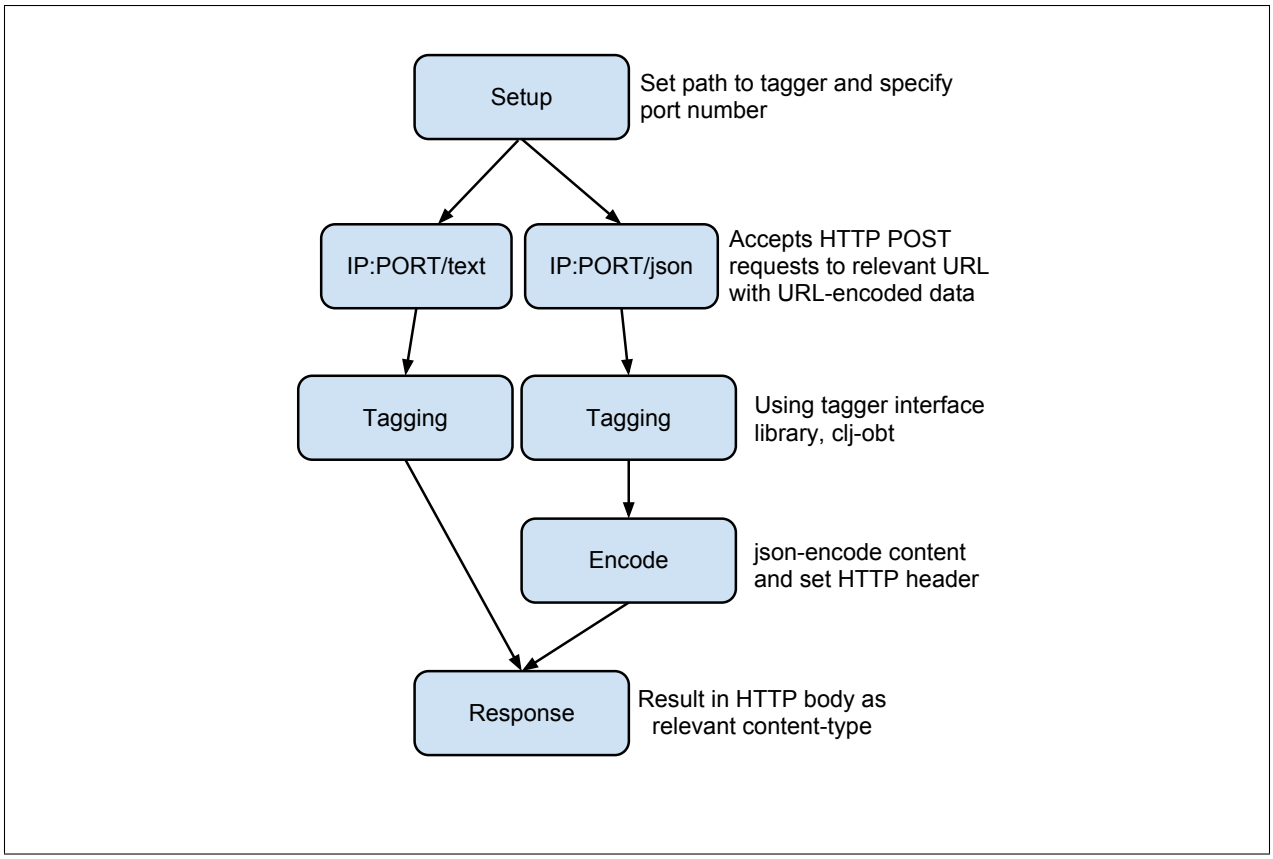

Figure 3: Tagger Web Service (clj-obt-service) System Model

4.3.4 Web Service Client

Namespace: `clj-egsiona.tagger-provider`

The client code for the tagger web service have been built directly into the prototype system. The optimal solution would be to include the client in a future version of the web service library, since these parts work in unison.

In order to select the desired functionality—web service or local program—one simply calls the appropriate function: `set-obt-program` or `set-obt-service`, providing the relevant location as an argument. The user will not need to select this manually, as the configuration functions of the prototype system will detect this automatically by examining the argument passed in. If the argument ends with a port number, the web service function will be used. If not, the local program function is used.

In contrast with a local installation of OBT, using the web service requires slightly more processing before and after the call to the tagger. This is negligible, as the largest source of delay will be latency. Neither this should be a large issue, as plain text does not consume much space. The server accepts of the `HTTP POST` requests, where the text to be tagged must be in the `data` parameter in the HTTP body. The content-type should be `application/x-www-form-urlencoded` and the service will return data with the encoding `ISO-8859-1`. The encoding is due to the output from The Oslo-Bergen-Tagger. The text that should be tagged must be URL encoded, which looks like the example in listing 13. Notice that the client does this automatically, and the user never need know the data was URL encoded—this is an implementation detail.

```
"Dette må tagges" => "Dette+m%C3%A5+tagges"
```

Listing 13: URL encoding of text to be tagged

The URL encoded text is used in the `POST` request. An example of a valid request can be seen in listing 14, with implementation details specific to the HTTP handling library used by the client. For example, the ISO encoding is specified, which makes the client assume the return data is in this format.

```
{:body "data=Dette+m%C3%A5+tagges"
 :content-type "application/x-www-form-urlencoded"
 :as "ISO-8859-1"}
```

Listing 14: Valid HTTP request to tagger service

The HTTP request in listing 14 is then sent to the appropriate URL using the `POST` request method. The URL will be in the format "`IP:PORT/node`" where the node is either `text` or `json`. The web service will then take the text from the data parameter, tag it, and send back a response looking like the data in listing 15.

```
{:trace-redirects ["http://localhost:8085/text"], :status 200,
 :headers {"date" "Tue, 13 Mar 2012 12:25:37 GMT",
           "connection" "close", "server" "Jetty(6.1.25)"},
 :body "[{:tags [\"pron\" \"nøyt\" \"ent\" \"pers\" \"3\"],
          :lemma \"dette\", :word \"Dette\", :i 1}
         {:tags [\"verb\" \"pres\" \"tr6\" \"pa4/til\"
                 \"<aux1/infinitiv>\"],
          :lemma \"måtte\", :word \"må\", :i 2}
         {:tags [\"verb\" \"pres\" \"inf\" \"pass\" \"tr1\"
                 \"<<<\"],
          :lemma \"tagge\", :word \"tagges\", :i 3}]"}
```

Listing 15: HTTP response from tagger web service

The body of the response in listing 15 (in the `:body` key) consists of one string, which is a identical to the output from the tagger. It is in the form of the Clojure data type *map*. In order to use this string as an actual object, it is transformed with the built-in *read-string* function. This function transforms the string from data to actual source code, making it usable programmatically as a native data structure, as seen in listing 16.

```
[{:tags ["pron" "nøyt" "ent" "pers" "3"],
  :lemma "dette", :word "Dette", :i 1}
 {:tags ["verb" "pres" "tr6" "pa4/til" "<aux1/infinitiv>"],
  :lemma "måtte", :word "må", :i 2}
 {:tags ["verb" "pres" "inf" "pass" "tr1" "<<<"],
  :lemma "tagge", :word "tagges", :i 3}]
```

Listing 16: Transformed HTTP response data into Clojure code

At this point, the client mirrored the tagger functionality, just as if the tagger was installed locally on the machine. This effort was a success and has enabled the prototype to become more platform independent.

4.3.5 The Extraction Software

Near the end of the development cycle, the prototype system was prepared for public release. It is made available at Github[17] and on the Clojars[18] repository.

The finalizing steps have mainly been cleaning up the prototype system, refactoring to expose a clean API, removing code no longer in use, and writing setup functions for simple configuration. Even though this project have been developed in relatively short time, compared to real life systems, it contained a surprisingly large amount of old code that had fallen out of use. This can probably be attributed to the exploratory nature of the project, with fast feedback from the corpus evaluation and the different techniques explored. The finalized state of the code base have had most of this unused code removed. This is mentioned shortly in the introduction to the development section, back in section 4, as some of the code discussed throughout the iterations is not present in the published library. This is code that is related to processing and conversion of the data worked with in the project, but have become irrelevant or obsolete.

In order to make the library simple to use, a readme file was written with explanation of the configuration and some examples of usage. At the very least, the user will need to configure the tagger software, either to the web service or a local installation. There are also two examples of database configurations, using PostgreSQL and SQLite. If a database is used, the tagger software will cache the tagged text. This improves the processing speed, if a given text have been tagged previously. In order to present a clean API to the user, some refactoring was done. An example of this was moving out most of the processing related code from the core namespace (`clj-egsiona.core`), into a new one: `clj-egsiona.processing`. The system model in figure 4 shows how the data flows through the prototype system, giving an overview of how it is used. Both the input text and the tagged text is used to find locations. Code related to the processing and disambiguation steps shown in the system model, will be found in the new `clj-egsiona.processing` namespace.

[17] https://github.com/ogrim/clj-egsiona
[18] https://clojars.org/clj-egsiona

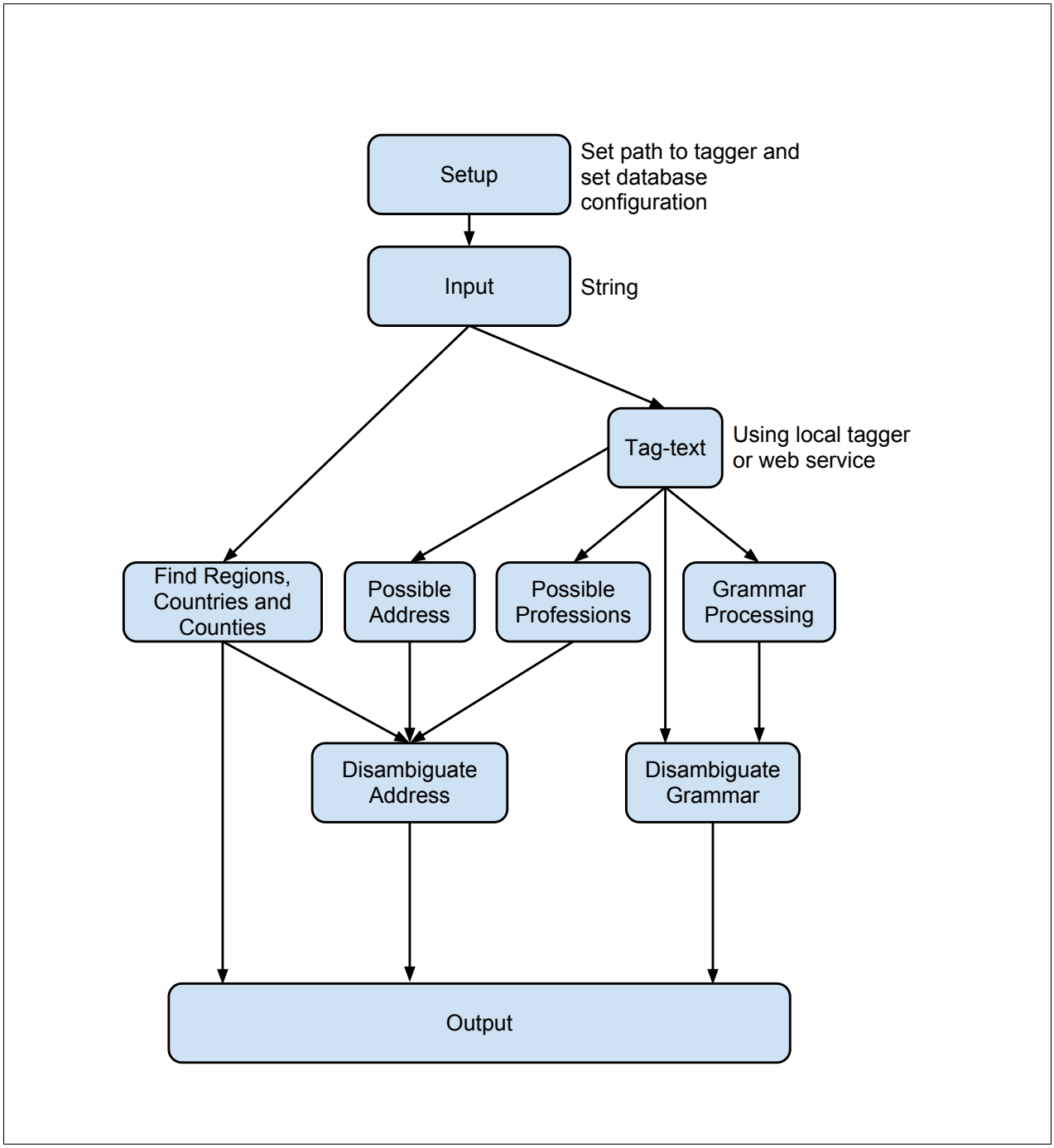

Figure 4: The Extraction Software (clj-egsiona) System Model

4.3.6 Dropping Proprietary Data Sets

The SSR data set have been used as a filtering device in the disambiguation process. It would prove problematic (and illegal) to release the prototype system with this data included, since it is proprietary. When making the prototype ready for public release, this had to be taken out of use. Interested in the current performance on the corpus, the evaluation was run one last time.

Without the SSR data set, the prototype system was able to score an f-measure of 76.2%, dropping only 0.6%. This was a pleasant surprise, as the prototype now may be of use without access to proprietary, high-grade data sets.

4.3.7 Evaluation

This iteration contained improvement of the prototype system's ability to find locations, as well as finalizing and exposing the project as a software library. Better grammatical processing was introduced to find candidate words. This made the rating a little worse, but improved the recall substantially. In order to bring up the precision, further work on disambiguation was done.

Multiple instances of entity recognition was implemented, the most concrete being domain name recognition and profession recognition. A more generic entity recognition was then implemented, which is a weakness in the disambiguation process, as it can be viewed as a form of over-specialization. The entities added here is inspired by the false-positives from the corpus evaluation, but an attempt to generalize this was made. After the new functionality with the grammatical disambiguation was implemented, the f-measure had risen to 76.8%.

Some work on tooling was resumed, which helped making the prototype system usable on more platforms. The tagger functionality was exposed as a web service, and client support was implemented in the prototype. The software then underwent some cleaning and refactoring, in order to be released to the public as a stand-alone library. Lastly, the use of proprietary data sets had to be removed. Discovering that the f-measure only dropped 0.6%, the prototype is usable without access to high-grade data sets.

5 Evaluation and Discussion

There are two distinct methods of evaluations used in this project: analytical and descriptive. The analytical evaluation is the dynamic analysis of the tagging accuracy, using metrics from information retrieval. The descriptive analysis constructs some scenarios around the prototype system, in order to demonstrate its utility. This was done by implementing a demo application, using the prototype system as a library. After reviewing the evaluations, there is a discussion of the findings and the implications.

5.1 Analytical Evaluation

Throughout the development phase, the prototype was evaluated by the IR techniques discussed in section 3.2. The metrics were collected, commented and put into table 5. The earliest metrics are from the second iteration, where the prototype system started taking shape and was able to find some locations. In the table, row 1 through 11 are measurements from the second iteration. This leaves only 4 measurements, all from the third iteration. There are significantly less data points from the third iteration, but this reflects what was implemented in the respective iterations.

The first iteration consisted mostly of tooling and data related work, with no evaluation performed. The second iteration consisted mostly of work relating to the core functionality of the prototype system, resulting in the largest amount of data points. The third iteration consisted of some work on core functionality, in addition to the web service library and preparing the prototype for release.

The very first data point in row 1, has very weak results. At this point, only capitalization and unwanted tags are filtered away. The recall is good, but the precision is terrible, resulting in a poor f-measure. From this very first measurement, there are lots of room for improvement. Already in row 2 the precision has been greatly improved. This is due to use of the proprietary SSR data set, with 73 671 unique locations in Hordaland. The recall suffers, as all locations not in Hordaland are filtered out. Still, the precision is greatly improved, giving a much improved f-measure.

n	Precision	Recall	F-measure	Summary
1	24.1	95.2	36.9	Filtering based on capitalization and simplistic grammatical processing
2	55.0	77.6	62.0	Using SSR data set
3	62.5	70.8	63.8	Name expansion
4	62.6	78.2	67.4	Finding middle names
5	63.1	82.1	69.1	Road detection
6	69.9	80.2	71.9	Handling first words in sentences
7	72.2	80.9	73.5	Handling professions in address recognition
8	72.7	80.8	73.7	Punctuation problems from OBT in names
9	72.7	81.2	73.9	Address validation
10	72.8	82.5	74.7	Handling sentences not split properly from OBT
11	73.1	82.5	74.9	Handling more punctuation in country.clj
12	64.5	93.3	73.0	Added grammar detection
13	66.5	94.0	75.0	Domain name and entity detection
14	69.1	93.6	76.8	Using entities in disambiguation
15	72.2	87.7	76.2	Removing proprietary data set (SSR)

Table 5: Evaluation by precision, recall and f-measure

The rest of the data points from the second iteration, row 3 through 11, are mainly not *very* interesting or problematic. The main focus was on improving precision, to gain a better f-measure. This was achieved in small steps, by implementing functionality regarding disambiguation and by fixing smaller problems with false-positives. Row 6 is probably the most interesting data point in this selection, even though it represents a small change. The precision sees quite an improvement from 63.1% to 69.9%, just by handling the first words in the sentences. The recall suffers from this, so the improvement in the f-measure is nearly 3%.

The remaining data points, 12 through 15, are from the third iteration. The change from row 11 to 12 is very interesting, as the f-measure gets worse. The change was kept in the prototype system, which at first glance may seem counterproductive. The change is kept due to the increased recall, at the cost of precision. This poorer result was accepted because there was potential for improving the precision by working with the disambiguation. If the recall is too low, there are limits to how much improvement in the f-measure

disambiguation can provide. In the rows 13 and 14, the precision was improved enough to take the f-measure past the slight drop, and to a high of 76.8%.

Row 15 is crucial, as it also accepts lowering the f-measure when removing use of the SSR data set. The prototype system was still able to perform acceptably without the proprietary SSR data set, which was unexpected. The importance of this is within social and legal aspects, discussed in section 3.3.2. The social aspects is the access to high-grade, possibly prohibitively expensive, proprietary data sets. Such data are not available for everyone. The other aspect is legal concerns, regarding the publishing of the prototype system with respect to the copyright of the proprietary data set. If the performance of the prototype system was dependent on the SSR data set, it would imply a barrier to implement a similar solution. It would also prohibit the redistribution of the prototype system as an open source project.

5.2 Descriptive Evaluation

As a method of evaluation within design science, descriptive evaluation is performed. These descriptive scenarios have been a part of the motivation for working on this project, and have been suggested as potential uses for the prototype system in section 1.3. In this section the scenario is first described, before evaluating it based on the performance of the prototype system.

5.2.1 Tag Generation

This usage is intended for writers and publishers, in order to help with location awareness in tag generation. Looking at the results from the analytical evaluation, where the prototype system is executing on the corpus, it is clear it does not yield perfect results. In row 15 in table 5 the prototype was able to find about 87% of the locations, with about 72% precision. This indicates that relying on the prototype system alone is likely to produce a fairly large number of faulty tags. Having a human user as a filter, can reduce the impact these tags. The manual labor involved in tagging text makes it a cumbersome task, which the prototype system can help alleviate.

Figure 5: Demo Application Text Input

When the user have written a blog post or a news article, the software can be used to automatically generate geographical tags. The user should be able to hit a single button, e.g. "find locations", which makes the prototype process the text and return detected locations. When the possible locations have been found, the user can select the appropriate tags simply by clicking on them. The user interface for this activity should be some form of buttons, in order to select and deselect all or some of the suggested tags. These operations should be implemented in order to support simplicity and efficiency. The user can then accept all suggested tags or only select—or deselect—a few tags, enabling the user to take the path of least resistance and be efficient. This frees the user from manually having to type in all the locations that should be tags, and provides quick means of selecting the appropriate tags. These specifications was implemented in a demo application, using the prototype system as a library.

The demo application have been implemented as a webapp using Clojure as the programming language. The demo application is runnable on the JVM, directly from the command line. It uses the geocoding code that was scrapped from the prototype system in section 4.2.4, in order to retrieve coordinates and place the locations on a map. The map is generated with the Google Maps JavaScript API, and requires an API key[1]. Three screenshots of the application in use have been enclosed (figure 5, 6 and 7), and the source code is available at Github[2] along with a runnable jar file[3]. The webapp is started with arguments specifying port number, tagger location and Google Maps API key. The command in listing 17 will start the webapp on port `8082`, using the tagger hosted at `localhost:8085` by `clj-obt-service` and will use an Google Maps API key generated for this thesis. The key in the listing will stay active as long as possible, so an attempt to use it can be made. If it does not work, please consult the Google Developer documentation to obtain a new key.

```
java -jar demoapp-1.0.0-SNAPSHOT-standalone.jar
      8082 localhost:8085 AIzaSyAH_DqRNhgdcTAR2jI_aQkPz6GC-qy7m7s
```

Listing 17: Running the demo application

In figure 5 the manual text input is being used. The demo application supports extracting content automatically from an URL, or from the input in the text area. There is a simple navigation menu on the top. The input text is processes by the prototype system upon clicking the "Process text" button. In figure 6 the article have been processed, and the resulting tags are marked up in the original text to make them stand out. All tags that were found are listed in the right column, with buttons easy selection. Four tags are selected, and these are highlighted in a different color in the original article. When hovering the mouse over the tags in the right column, the corresponding words in the article light up as well. This makes it easier to find the correct tags within the text. When the user is happy with the tag selection, the button "Save selection" is clicked. This will save the data, and display the article text with the selected tags only. A map view is shown with markers on the corresponding locations. This can be seen in figure 7.

[1] https://developers.google.com/maps/documentation/javascript/tutorial#api_key

[2] https://github.com/ogrim/clj-egsiona-demoapp

[3] https://github.com/downloads/ogrim/clj-egsiona-demoapp/demoapp-1.0.0-SNAPSHOT-standalone.jar

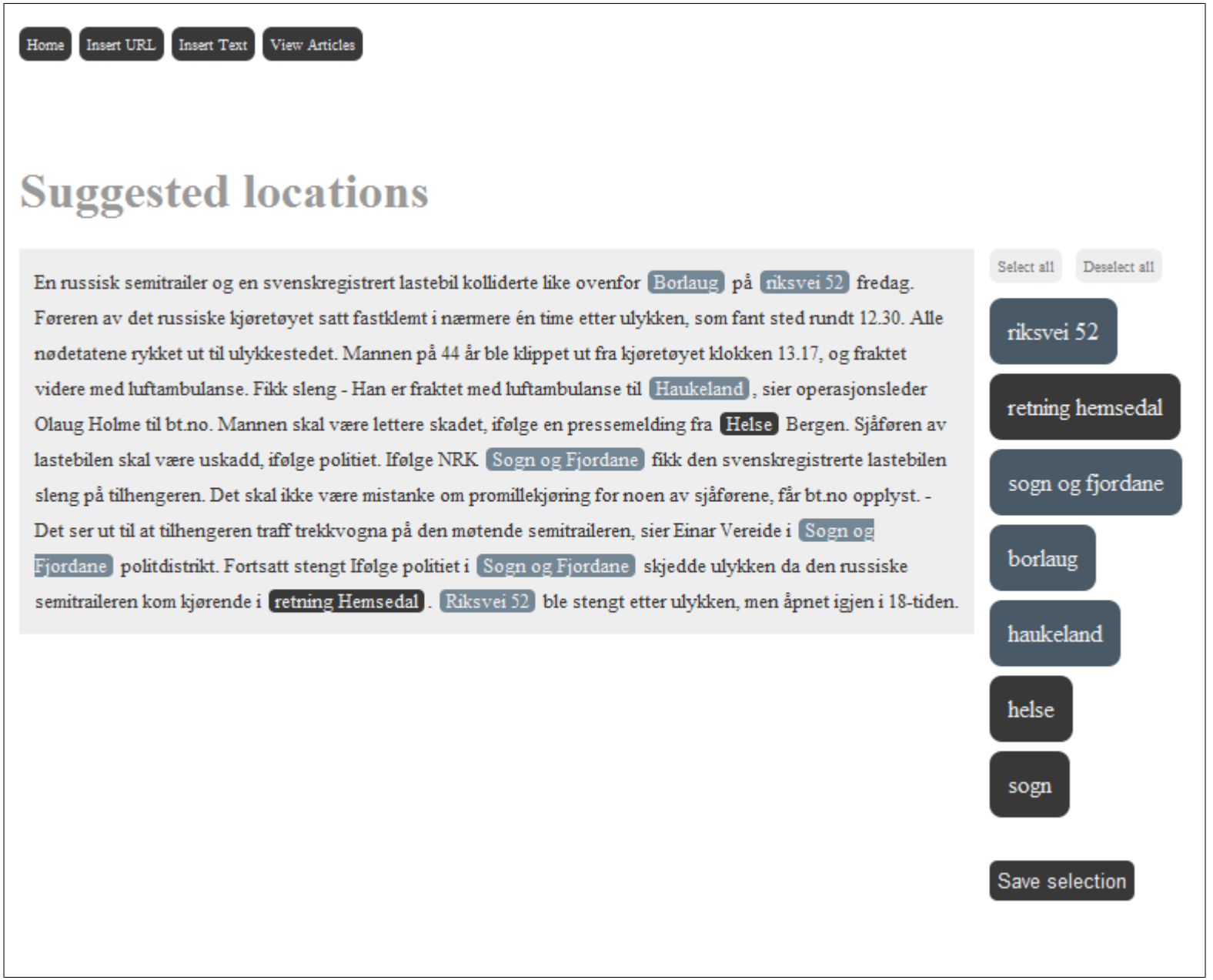

Figure 6: Demo Application Tag Selection

Figure 7: Demo Application Article View

To discuss the problems with the use of the prototype system for tag generation, the article used in the demo application figures will be examined. This article is available from BT.no[4]. "*Han er fraktet med luftambulanse til Haukeland*" (he was transported by air ambulance to Haukeland) is referring to Haukeland as a proper noun, as it is a hospital in the region. Similarly, the following sentence reads "*(..) ifølge en pressemelding fra Helse Bergen*" (according to a press release from Bergen Hospital Trust). The prototype system have tagged "Helse" as a location, but it is part of the proper noun "Helse Bergen". In both these examples, the proper nouns overlap in meaning with location names. This classification can be ambiguous even for humans. In a very strict sense, the words should not be tagged as locations since they are part of a proper noun. On the other hand, the location in the proper noun still informs the reader to an actual location that is relevant to the article. While reading about the Haukeland Hospital, the reader will probably know where it is located—which provides the reader with some geographical semantics.

Another problem is in subdivisions of possible locations. The prototype system is suggesting both "*Sogn og Fjordane*" and "*Sogn*" as possible locations. The demo application have not highlighted "*Sogn*" in the article, as the word is contained within "*Sogn og Fjordane*" and already have been highlighted. The demo application will highlight the words in the sequence it gets the locations from the prototype system, and does not allow overlapping highlights. Since "*Sogn og Fjordane*" is in front of "*Sogn*", this is highlighted first. It is possible to only save the "*Sogn*" tag, which then would highlight this instead. These aspects are problems—or features—in the demo application, while the prototype system is the real offender. The prototype system could possibly be extended with some location name overlap detection, with the goal to increase tagging precision.

Another related problem is the "*retning hemsedal*" tag. Hemsedal is an actual location, but it have been grouped together with the noun "direction". The direction is not a part of the location name, making this an incorrectly tagged word. The problem is really in the grouping, as the prototype system was able to find the location Hemsedal, but grouped it together with a wrong word. A possible solution can be implemented in the demo application: functionality for manually splitting and selecting subdivisions of the suggested tags it gets from the prototype system.

[4]http://www.bt.no/nyheter/lokalt/n-fastklemt-i-ulykke-2698455.html

In this demo application the erroneous tags are not a big problem, as it is very easy to select the correct ones. The largest issue is with incorrect grouping of tags, and in adding new locations not found by the prototype system. These problems can be fixed in the demo application, rather than the library providing the tags. Since there is a human user which selects the appropriate location tags, this usage of the prototype system seem to be a good match. The impact of the precision problems in the prototype is reduced due to the human user, while providing utility and value.

5.2.2 Construction of External Tools

There are different uses for the prototype system in the construction of various external tools. One example of this is within the use of existing frameworks, like the Stanford NLP. Using the prototype system to generate training data, it can be possible to construct a generalized location recognizer with the Stanford NLP. This would at best perform equally well as the prototype system, but the most likely outcome is worse performance. Using data from one region, the recognizer could be generalized to perform better on articles from another region.

One concern is the performance of the prototype system on online news articles published in counties other than Hordaland. This was especially a concern while using the proprietary SSR data set, with location names in Hordaland. After removing the SSR from use, the performance dropped slightly, which indicates the dependency on the SSR data set was not very strong. If the performance of the prototype system was dependent on external data sets, it is likely it would perform worse on articles from another region. Because we already have an acceptable result without external data, constructing a generalized location recognizer is probably *not* a very fruitful avenue to explore further. It seems likely that the prototype can perform to an acceptable level on articles from other regions as well.

There is another external tool that could be constructed from the prototype system. One aspect that currently only is exploited internally, is the functions used for disambiguation. The software is able to do name, address, profession and some entity recognition. This information is never returned to the user, although this is something it very well could. By using these functionalities, one could use the relevant parts of the prototype system to

serve as a foundation to implement a primitive named entity recognizer. Throughout the literature reviewed, a NER was commonly used in the different projects. An open source NER that supports Norwegian, would be a great asset to both developers and researchers alike.

5.2.3 Semantic Applications

This use case is closely related to tag generation, only in a more formal sense. The main difference is in the user interface, and additional processing regarding the use of ontologies. Given a specific ontology, the software can be used for generating RDF statements about the news articles it processes. In the prototype's current working condition, this use case would generate a lot of faulty statements. Based on the domain in which the generated RDF statements would be used, one could apply a human censor to select only the statements deemed correct or necessary. The activity of selecting tags should, as in the tag generation case, be available through a suitable graphical user interface.

Considering the discussion in the previous section, regarding named entity recognition, there are opportunities to use the prototype in additional semantic fashions. If a NER is developed from the prototype system, this functionality could be used to add other kinds of semantic statements about the articles in addition to only the locations. In combination with a POS tagger to use grammatical processing, it could be possible to derive additional semantics regarding the content of the news articles. One approach would be to construct a NER that detects personal names and quotes, and use grammatical processing with the POS data in order to generate statements like "who said what" and more. This could make the prototype system more powerful in uncovering additional semantics regarding locations. An example would be to use the POS data to find adjectives and nouns describing a location, and generate statements like "location X *is a* fun place".

5.2.4 Statistics and Metrics

Another possible use of the prototype system is within generation of statistics and metrics. For this to operate autonomously, the article collector is required to be a more integrated part of the prototype system. It could also stand to be improved, which was

discussed in the evaluation of the first iteration (section 4.1.7). To achieve an autonomous system to generate statistics over time, the prototype would have to be integrated with the article collector by some logic to handle the scheduling of data collection and processing.

The main problem in this scenario is the quality of the data the system would generate. The prototype's f-measure on the corpus is about 76%, which most likely yields too noisy data for many of the possible use cases for metrics. In environments where noisy data can be handled, the software might be appropriate for use. One could perhaps also say something akin to "*We know X with 76% accuracy*" when using data from this software, if such an answer is good enough.

If the prototype system is set to process different regional news sites, it is possible to get metrics and find different clusters. A possible use is applying an algorithm to determine the different online news sites' geographical focus. The algorithm to determine geographical focus can be inspired by the survey of previous work performed by Amitay et al. (2004) and by the algorithm they develop. It would be possible to use the prototype in combination with another analysis method, in order to uncover patterns in the articles. An example of this applying sentiment mining to the articles. Sentiment mining finds *negative* and *positive* words, and classifies the text according to its sentiment. An example of this is by Leetaru (2011) who was able to find a trend towards negative more negativity in the press, before the Arab Spring revolutions. Using sentiment mining in combination with the prototype system, it would be possible to uncover negative or positive bias towards particular locations, or by particular online news papers.

5.3 Discussion

With the evaluations in mind, the success criteria and research question will be revisited and discussed to determine the level of success. The research question asked "*how can we automatically detect geographical information in online news articles?*" with four success criteria. Before answering the research question, the success criteria are examined.

1. detect possible locations by text analysis
2. represent complex[5] locations
3. create mappings between an article and locations
4. provide accurate data while minimizing false positives

The first success criterion is detecting possible locations by text analysis. The whole implementation of the prototype system does text analysis to find locations. Some of the problems faced was use of external data sets, grammatical processing and disambiguating the possible locations. The data used was at first proprietary, but this was removed without a large drop in accuracy. In the end, the prototype system was only using public data, like countries and municipality names, in addition to lists of personal names. The various components that constitutes the prototype system have been discussed throughout the iterations, by referring to namespaces, libraries and programs. An important external program have been the Oslo-Bergen-Tagger, which performs part-of-speech tagging on the input text. Without this resource, grammatical processing would be a lot more problematic. The analytical evaluation shows the prototype is able to detect the locations, as it found 87.7% of the locations in the corpus. Along with the demo application, this functionality have been demonstrated successfully. The first success criterion appears to be fulfilled.

The second success criterion is representing complex locations, which are non-primitive shapes like polygons. At the start of the project, high-grade geospatial data was attained, and much effort went into applying it to the prototype system. Some tools were made to process and import the data into the PostGIS database. The first batch of data did not contain any location names, and more data was requested. After gaining access to the SSR data set, the distinctly Norwegian SOSI file format had to be parsed and imported to the database. Again, work was done in tooling to support this data type. At this point, the prototype had location names paired with coordinates. However, coordinates are not complex locations. The second criterion defines a complex location as one represented by non-primitive shapes, like a polygon. The shapefiles first received are complex locations by this definition.

[5]Locations represented by non-primitive shapes like polygons

Simply by using a geospatial database like PostGIS with high-grade map data, the prototype was able to represent complex locations. This seems to fulfill the criterion, but only in a superficial sense. As it was discussed in section 4.1.6, the use of the high-grade geospatial made little progress around the second iteration. This was largely due to lack of experience with geographical data, and inability to solve the problems faced. This ultimately led to dropping the use of high-grade geospatial data in the prototype system. One can argue the second criterion is fulfilled by technology choice and data access alone, but considering that use of this fell out during the second iteration, the argument is rendered moot. The prototype system have not gained greater utility from this technology, indicating failure to fulfill this criterion.

The third success criterion is creating mappings between an article and locations. This have been trivial to achieve, since the locations are not complex shapes. The prototype system creates the mapping by returning the found locations as plain text, which is a rather loose mapping. The way the prototype system was used by the demo application, discussed in section 5.2.1, shows the demo application using the plain text locations in order to create the mappings. It highlights the relevant words in the original article, using simple text comparison. Without complex locations, creating mappings seem only relevant for the application that is using the data from the prototype system. Had the locations been represented by complex shapes, the mapping would have been more relevant for the prototype system as well. The third criterion seem fulfilled to a sufficient extent, even though the meaningfulness of this criterion is reduced without the usage of complex map data.

The fourth success criterion is providing accurate data while minimizing false positives. This have been measured using metrics from information retrieval. In order to perform the evaluation, a corpus was collected and tagged manually. After implementing the metrics in code, the evaluation could be run by calling a single function. The evaluation was run after implementing new functionality, and was used as an indication if a change or functionality was worth keeping. At the end of the third iteration, the f-measure was about 76%. A better score had been hoped for, but development had to stop due to time constraints. Smaller fixes could not help the score, without stooping to over-specialization. The entity recognition already comes somewhat close to this. The fourth success criterion seem fulfilled to some extent, considering the IR metrics achieved.

Having reviewed the different evaluations of the prototype system, there are both positive and negative results. It is positive that the prototype system is not tied to a proprietary data set, since the SSR data set was removed from use. This seems to indicate anyone can implement a similar solution without prohibitively expensive data sets. The final libraries and the tagger web service are also positive results, as they proved very useful in implementation. The libraries have helped with greater platform independence and with implementing the demo application.

The results that are more negative, is the current level of grammatical processing. I expect the prototype system will perform better with help from proper computational linguists. The most negative result was the failing to use the geospatial data, which lead to failing to fulfill the second success criterion, and diminished the relevance of the third. The prototype system would probably have different performance characteristics, if time was not lost to working with geographical data that did not provide any utility.

With all four success criterion discussed, the main research question is at hand. This entire thesis have documented the implementation of a prototype system able to detect geographical information in online news articles. The system is usable, as demonstrated with descriptive evaluation and the demo application. Using analytical evaluation, the information retrieval metrics have shown the systems performance on the corpus. Only the second success criterion have failed completely, while the third lost much of its relevance as a result of this. The first and fourth success criterion appear to be fulfilled to a satisfactory degree. In all, these factors seem to indicate the thesis is able to successfully answer the research question of how to automatically detect geographical information in online news articles. There are also some limitations, that needs to be considered.

Given that the prototype was developed with a corpus of online news articles published in the county of Hordaland, the performance should be re-evaluated on an entirely new corpus, published in another region. The analytical evaluation gave an f-measure of 76.2%, but this number will surely fluctuate when evaluating a new corpus. The deviation should not be to great on a similar corpus, a new set of online news articles published in Hordaland county. If the f-measure decreases substantially, it is an indication that the prototype system have been over-specialized to the current corpus. The prototype should also be evaluated on a more dissimilar corpus, preferably one consisting of online news articles published in an entirely different region of Norway.

6 Summary and Conclusion

This thesis documents the development and implementation process of a prototype system that finds locations in online news articles. At first, high-grade map data was attempted to be used. The importance of geospatial data is reflected in the success criteria, and the work done with tooling to support it. The largest problem in the project have been in the application of such data. The effort stalled and failed, and the high-grade map data was dropped in the second iteration.

There have been successful work in developing libraries, that have been used in the implementation of the prototype system. This is the interface to the Oslo-Bergen-Tagger and the tagger web service. The prototype itself has also been exposed as a library.

Large parts of the prototype system was developed in the second iteration, with much positive result. The tagging performance was gradually increased through most of the iterations. At end of the third iteration, the SSR data set was taken out of use without a substantial drop in performance. A demo application was developed using the prototype system as a library, which was used for descriptive evaluation.

6.1 Conclusion

There were several positive results, such as acceptable tagger performance in the prototype. Another positive result was the removal of the proprietary SSR data set, which degraded the performance less than expected.

There are also negative results from the project. The problem that had the most impact, was the failure to use high-grade map data. Another problem is the possible overspecialization with the entity recognition, used in the disambiguation. The tagging performance, as measured by the analytic evaluation, is acceptable but could also be improved.

The end result is restricted by the analytical evaluation, as the prototype was not able to find all locations and had a limited precision. The prototype was still able to demonstrate

its applicability with the descriptive evaluation, particularly in the demo application. The prototype system has limitations, but was nevertheless demonstrated to be useful.

The success criteria have been discussed, and there were varying results. The prototype system is able to detect possible locations by text analysis, create mappings between an article and locations, and to a certain extend provide accurate data while minimizing false positives. Only the second success criterion failed completely, which also reduces the relevance of creating mappings between articles and locations. The accuracy is measured analytically with information retrieval metrics, and the result was an acceptable f-measure of 76.2%.

The research question asked was "*how can we automatically detect geographical information in online news articles?*" Having discussed the different results, the findings seem to indicate the thesis is able to successfully answer the research question. I therefore conclude that the prototype system is able to extract geographical entities from online news articles, published in Hordaland.

6.2 Further Work

Due to the failure to utilize the high-grade map data from Norge Digitalt, further work in this avenue can be interesting. With help from domain specialists, like geographers, the prototype can be extended to use more geospatial processing. One immediate avenue of interest is in improvement of the disambiguation functionality. When the prototype system find a location, it could be resolved to a coordinate pair using geocoding or high-grade map data, in combination with geographical processing to determine the correct location. Other higher geographical functionalities could also be explored, like the handling of roads and directions.

A criticism of the prototype system is the limited use of grammatical processing. It is probable computational linguists will have much to contribute in this area. There might exist tools and techniques that would help in the prototype. In the literature section, the use of named entity recognition was used in several projects. The prototype system could benefit greatly from a robust named entity recognition, which should be constructed by proper computational linguists. Greater knowledge of grammar and language would also

help in improving the selection of words that possibly are locations. This functionality is currently only using limited grammatical processing.

References

Adida, B., S. Annotation, C. Commons, S. Francisco, et al. (2011, April). *Handbook of Semantic Web Technologies*. Springer.

Amitay, E., N. Har'El, R. Sivan, and A. Soffer (2004). Web-a-where: geotagging web content. In *Proceedings of the 27th annual international ACM SIGIR conference on Research and development in information retrieval*, pp. 273–280. ACM.

Andersen, R. (2009). About us. http://www.statkart.no/eng/Norwegian_Mapping_Authority/. Accessed: 17/10/2011.

Bizer, C., T. Heath, K. Idehen, and T. Berners-Lee (2008). Linked data on the web (ldow2008). In *Proceeding of the 17th international conference on World Wide Web*, WWW '08, New York, NY, USA, pp. 1265–1266. ACM.

Brenna, A. (2008). Regjeringen sletter norge fra kartet. http://www.digi.no/790893/regjeringen-sletter-norge-fra-kartet. Accessed: 09/12/2011.

Eclipse Foundation (2011). Eclipse public license 1.0. http://www.eclipse.org/legal/epl-v10.html. Accessed: 09/05/2011.

Esri (2009). Geoprocessing considerations for shapefile output. http://webhelp.esri.com/arcgisdesktop/9.3/index.cfm?TopicName=Geoprocessing%20considerations%20for%20shapefile%20output. Accessed: 17/10/2011.

Fink, C., C. Piatko, J. Mayfield, D. Chou, T. Finin, and J. Martineau (2009). The Geolocation of Web Logs from Textual Clues. In *Computational Science and Engineering, 2009. CSE'09. International Conference on*, Volume 4, pp. 1088–1092. IEEE.

GATE (2012). Non-english langugage support. http://gate.ac.uk/sale/tao/splitch15.html#x20-39000015. Accessed: 13/02/2012.

GeoNames (2011). Geonames. http://www.geonames.org/. Accessed: 10/05/2011.

Google (2011). The google geocoding api. http://code.google.com/apis/maps/documentation/geocoding/. Accessed: 09/12/2011.

Hevner, A., S. March, J. Park, and S. Ram (2004). Design science in information systems research. *Mis Quarterly 28*(1), 75–105.

Ide, N. and J. Véronis (1998, March). Introduction to the special issue on word sense disambiguation: the state of the art. *Comput. Linguist. 24*(1), 2–40.

Kartverket (2011). Norway digital - the national geographical infrastructure. http://www.statkart.no/Norge_digitalt/Engelsk/About_Norway_Digital/. Accessed: 09/12/2011.

Leetaru, K. H. (2011). Culturomics 2.0: Forecasting large-scale human behavior using global news media tone in time and space. *First Monday 16*(9-5).

Manning, C., P. Raghavan, and H. Schütze (2008). *Introduction to information retrieval*, Volume 1. Cambridge University Press Cambridge, UK.

Michael D. Lieberman, Hanan Samet, J. S. . J. S. (2007). Steward: architecture of a spatio-textual search engine. In *Proceedings of the 15th annual ACM international symposium on Advances in geographic information systems*, pp. 25. ACM.

Noer, L. K. (2008). Rådyre kart fra staten. http://www.nrk.no/helse-forbruk-og-livsstil/1.5824212. Accessed: 09/12/2011.

Nunamaker Jr, J. and M. Chen (1991). Systems development in information systems research. In *System Sciences, 1990., Proceedings of the Twenty-Third Annual Hawaii International Conference on*, Volume 3, pp. 631–640. IEEE.

Parr, T. (2007). *The Definitive ANTLR Reference: Building Domain-specific Languages*. Raleigh: Pragmatic Bookshelf.

Solstad, Ø. (2009). Kartverket frigir kartene sine. http://nrkbeta.no/2009/11/12/kartverket-frigir-kartene-sine/. Accessed: 09/12/2011.

The Stanford Natural Language Processing Group (2012). Stanford log-linear part-of-speech tagger. http://nlp.stanford.edu/software/tagger.shtml. Accessed: 13/02/2012.

UniComputing, U. . (2012). The oslo-bergen tagger. http://www.tekstlab.uio.no/obt-ny/english/index.html. Accessed: 13/02/2012.

Van Rijsbergen, C. J. (1979). *Information retrieval*. London: Butterworths.

Wirth, N. (1976). *Algorithms + Data Structures=programs*. Prentice Hall.

WorldGazetteer (2011). World gazetteer. `http://www.world-gazetteer.com/`. Accessed: 10/05/2011.

Zong, W., D. Wu, A. Sun, E.-P. Lim, and D. H.-L. Goh (2005). On assigning place names to geography related web pages. In *Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries*, JCDL '05, New York, NY, USA, pp. 354–362. ACM.

Appendices

Appendix A List of Acronyms

This list contains an overview over the acronyms that are used throughout this text. Before an acronym is used, it is written out completely and explained if needed. For example, POS is explained, but XML is not.

Acronym	Definition
ANTLR	ANother Tool for Language Recognition
API	Application Programming Interface
BNF	Backus Naur Form
CLR	Common Language Runtime
DOM	Document Object Model
HTML	HyperText Markup Language
IR	Information Retrieval
JVM	Java Virtual Machine
NER	Named-Entity Recognizer
NLP	Natural Language Processing
OBT	Oslo-Bergen-Tagger
POS	Part-of-Speech tagger
RDF	Resource Description Framework
SOSI	Samordnet Opplegg for Stedfestet Informasjon (Coordinated Approach for Spatial Information)
SSB	Statistisk Sentralbyrå (Statistics Norway)
SSR	Sentralt Stedsnavnregister (Central Place Name Register)
TLD	Top-level domain
URL	Uniform Resource Locator
XML	Extensible Markup Language